

# Learning Natural Language Syntax

Stephen Watkinson

This thesis is submitted in partial fulfilment of the  
requirements for the degree of  
Doctor of Philosophy.

University of York  
York  
YO10 5DD  
UK

Department of Computer Science

April 2002

# Abstract

In this thesis, a system (CLL), is developed for learning Categorical Grammars for English. CLL has unsupervised and weakly supervised settings and is designed to learn from positive examples only. Such a setting is motivated by the fact that most data for natural language appears in this form. The approach is therefore seeking to be practical and may, in time, lead to more psychologically plausible models for the machine learning of natural language syntax.

CLL was built with the aim of using simple statistical methods and simple compression methods to extend symbolic syntactic constraints on the grammar learning problem. A stochastic version of Categorical Grammar is used to determine a set of the best analyses for natural-language examples. A lexicon is built from these analyses with the further constraint that the most compressive grammars are considered to be the best. Two methods of compression are investigated.

CLL has been extensively tested on a variety of corpora. Early experiments used simple, generated corpora that contained examples of a very small subset of English. However, later experiments were performed on more realistic examples extracted from the Wall Street Journal section of the Penn Treebank.

The results, while not of the quality of supervised techniques, do indicate that CLL is effectively learning natural language syntax, although further work to extend the coverage and accuracy of the system is suggested.

# Contents

<b>1</b>	<b>Introduction</b>	<b>12</b>
1.1	General Issues . . . . .	13
1.2	Why Learn Natural-Language Syntax? . . . . .	14
1.3	The Syntactic Model . . . . .	15
1.4	The Machine-Learning Model . . . . .	16
1.5	A Summary of the Learning Model . . . . .	20
1.6	Investigating the Model . . . . .	20
1.7	Questions Addressed in the Thesis . . . . .	21
1.8	The Contributions . . . . .	22
1.9	Conclusion . . . . .	23
<b>2</b>	<b>Grammatical Formalisms</b>	<b>25</b>
2.1	Requirements . . . . .	25
2.1.1	Expressiveness . . . . .	25
2.1.2	Learning Problem Issues . . . . .	26
2.1.3	Extensibility . . . . .	26
2.1.4	Separation of Knowledge . . . . .	26
2.1.5	Elegance . . . . .	26
2.2	Alternatives . . . . .	27
2.2.1	Lexicalised . . . . .	27
2.2.2	Stochastic Formalisms . . . . .	29
2.2.3	Unification Formalisms . . . . .	30
2.2.4	Selecting the Grammar Formalism . . . . .	32
2.3	Categorial Grammar . . . . .	33
2.4	Stochastic Models with Categorial Grammar . . . . .	35
2.4.1	A Language Model . . . . .	36
2.5	Conclusion . . . . .	43
<b>3</b>	<b>A Computational Framework for Syntax Learning</b>	<b>44</b>
3.1	A Computational Syntax-Learning Model . . . . .	44
3.1.1	Examples . . . . .	44
3.1.2	The Search Engine . . . . .	46
3.1.3	The Categorial Grammar Lexicon . . . . .	48
3.1.4	The Search Constraints . . . . .	48

3.2	General Implementation Issues . . . . .	56
3.2.1	Symbolic Learning . . . . .	57
3.2.2	Machine Learning Issues . . . . .	57
3.2.3	Stochastic Mechanisms . . . . .	61
3.3	Computational Language Learning Conclusions . . . . .	63
3.3.1	The Problem . . . . .	63
3.3.2	The Type of System . . . . .	63
<b>4</b>	<b>Computational Syntax-Learning Systems</b>	<b>65</b>
4.1	Part-of-Speech Tagging . . . . .	66
4.1.1	Supervised Part-of-Speech Tagging . . . . .	66
4.1.2	Unsupervised Part-of-Speech Tagging . . . . .	79
4.1.3	Supertagging . . . . .	82
4.2	Syntax Learning . . . . .	84
4.2.1	Supervised Syntax Learning . . . . .	84
4.2.2	Unsupervised Syntax Learning . . . . .	90
4.3	Conclusions . . . . .	96
<b>5</b>	<b>The Learning System</b>	<b>97</b>
5.1	Implementing the Computational Model . . . . .	98
5.2	The Input . . . . .	99
5.2.1	The Corpus . . . . .	99
5.2.2	The Rules . . . . .	101
5.2.3	The Categories . . . . .	102
5.2.4	The Initial Lexicon . . . . .	103
5.3	The CLL System . . . . .	104
5.3.1	The Basic Algorithm . . . . .	104
5.3.2	The Modules . . . . .	106
5.3.3	Properties of the Algorithm . . . . .	117
5.4	The Output . . . . .	119
5.5	A Worked Example . . . . .	119
5.6	Conclusions . . . . .	122
<b>6</b>	<b>The Parser</b>	<b>124</b>
6.1	The Centrality of the Parser . . . . .	124
6.2	The Requirements for the Parser . . . . .	125
6.2.1	The Computational Effectiveness of the Parser . . . . .	125
6.2.2	The Linguistic Appropriateness of the Parser . . . . .	126
6.2.3	The Psychological Plausibility of the Parser . . . . .	126
6.3	General Parsing Approaches . . . . .	126
6.3.1	Parse Order . . . . .	127
6.3.2	Top-Down and Bottom-Up Parsing . . . . .	127
6.3.3	Efficient Parsing . . . . .	129
6.3.4	The Basic Parsing Algorithm . . . . .	129



6.4	Approaches to Selecting Most-Likely Parses . . . . .	130
6.4.1	Determinism . . . . .	130
6.4.2	Stochastic Methods . . . . .	131
6.5	Conclusions . . . . .	131
7	<b>Human Language Learning</b>	<b>133</b>
7.1	The Examples . . . . .	133
7.2	The Background Knowledge . . . . .	135
7.3	Learning Mechanisms . . . . .	137
7.4	Summary . . . . .	140
8	<b>The Corpora</b>	<b>141</b>
8.1	Generated Corpora . . . . .	141
8.1.1	Simple Generated Corpora . . . . .	141
8.1.2	The LLL Corpus . . . . .	143
8.2	Naturally Occurring Corpora . . . . .	143
8.2.1	The Penn Treebank . . . . .	143
8.2.2	The Extraction Procedure . . . . .	146
8.2.3	The Extracted Corpora . . . . .	150
8.3	Discussion . . . . .	151
9	<b>The Experiments</b>	<b>153</b>
9.1	Experimental Parameters . . . . .	153
9.1.1	Initial Lexicon . . . . .	153
9.1.2	Parser . . . . .	154
9.1.3	Compression Metric . . . . .	155
9.1.4	CG Category Databases . . . . .	156
9.1.5	Noun-Phrase Handling . . . . .	156
9.1.6	Corpora . . . . .	157
9.2	Initial Experiments . . . . .	157
9.3	Large Scale Experiments . . . . .	157
9.4	Future Experiments . . . . .	159
9.5	Conclusions . . . . .	159
10	<b>Evaluation Methods</b>	<b>160</b>
10.1	The Results and Their Format . . . . .	160
10.2	Evaluation Techniques and Metrics . . . . .	161
10.2.1	Evaluation of the Early Experiments . . . . .	162
10.2.2	Lexicon Size . . . . .	162
10.2.3	Lexicon Ambiguity . . . . .	163
10.2.4	Lexical Accuracy . . . . .	163
10.2.5	Structural Accuracy . . . . .	163
10.3	Applying the Metrics . . . . .	165
10.4	Translation of the Penn Treebank Annotation . . . . .	165

10.4.1	Previous Work on Corpus Translation . . . . .	166
10.4.2	The Task . . . . .	167
10.4.3	Alternative Approaches . . . . .	168
10.4.4	Results . . . . .	171
10.4.5	Corpus Translation Conclusions . . . . .	173
10.5	Evaluation Conclusions . . . . .	173
<b>11</b>	<b>Results</b>	<b>175</b>
11.1	Initial Results . . . . .	175
11.2	Results for PC1 . . . . .	177
11.3	Results for PC2 . . . . .	179
11.4	Results for PC3 . . . . .	182
11.5	Results for PC4 . . . . .	185
11.6	Cross-Corpora Comparisons . . . . .	186
11.6.1	The Corpus . . . . .	186
11.6.2	The Initial Lexicon . . . . .	187
11.6.3	The Fixed Closed-Class Lexicon . . . . .	187
11.6.4	The Category Database . . . . .	188
11.6.5	The Beam . . . . .	188
11.6.6	The Compression Metric . . . . .	189
11.6.7	The NP Rule . . . . .	189
11.6.8	Experiment Length . . . . .	189
11.6.9	Experiment Progression . . . . .	190
11.7	System Comparisons . . . . .	193
11.8	Conclusions . . . . .	195
<b>12</b>	<b>Conclusions</b>	<b>196</b>
12.1	A General Evaluation of the Categorical Lexicon Learner . . . . .	196
12.2	Evaluating the General Contributions . . . . .	197
12.3	Further Work . . . . .	199
12.3.1	Developing the System . . . . .	199
12.3.2	Further Experimentation . . . . .	200
12.3.3	Evaluation . . . . .	200
12.4	Conclusions . . . . .	201
<b>A</b>	<b>The Penn Part-of-Speech Annotation</b>	<b>202</b>
<b>B</b>	<b>Initial Lexicons</b>	<b>204</b>
B.1	CCW1 . . . . .	204
B.2	CCW2 . . . . .	205
<b>C</b>	<b>The Category Databases</b>	<b>215</b>
C.1	CATDB1 . . . . .	215
C.2	CATDB2 . . . . .	215
C.3	CATDB3 . . . . .	217

# List of Tables

2.1	A set of example CG categories . . . . .	34
3.1	Closed-class word statistics for PC2 . . . . .	52
4.1	Example of the calculation of the most probable sequence of states . . . . .	76
4.2	Examples of the figure calculated by the forward-backward algorithm . . . . .	78
4.3	An example cue-based learning look-up table . . . . .	93
5.1	Summary of the corpora used with CLL . . . . .	101
5.2	Example categories and roles in the Scheme representation . . . . .	103
8.1	The functional labels in the Penn Treebank II . . . . .	144
8.2	A summary of the naturally occurring corpora used in the experiments . . . . .	151
9.1	The experiment plan when an initial lexicon is used . . . . .	158
9.2	The experiment plan when an initial lexicon is not used . . . . .	158
10.1	Table of category and lexicon information on the translated corpora . . . . .	172
10.2	Table of the category frequencies for both approaches . . . . .	172
10.3	Frequencies of words appearing in a frequency range of number of categories .	173
10.4	Table comparing the coverage of the two approaches . . . . .	173
11.1	Initial lexical and parse accuracy results . . . . .	176
11.2	Unseen example parsing accuracy . . . . .	177
11.3	A summary of the results using the PC1 corpus . . . . .	177
11.4	The baseline crossing-brackets rate results for PC1 . . . . .	178
11.5	The baseline lexical-accuracy values for PC1 . . . . .	178
11.6	A summary of the results using the PC2 corpus . . . . .	180
11.7	The baseline crossing-brackets rate results for PC2 . . . . .	180
11.8	The baseline lexical-accuracy values for PC2 . . . . .	180
11.9	A summary of the results using the PC3 corpus without an initial lexicon . .	182
11.10	A summary of the results using the PC3 corpus with an initial lexicon . . . .	183
11.11	A summary of the results using the PC4 corpus . . . . .	185
11.12	The baseline crossing-brackets rate results for PC4 without initial lexicon . .	185
11.13	The baseline lexical-accuracy values for PC4 . . . . .	185



# List of Figures

1.1	The computational learning model . . . . .	20
2.1	An example feature structure . . . . .	31
2.2	An example CG parse . . . . .	34
2.3	An example CFPSG parse . . . . .	35
2.4	An example CFPSG . . . . .	35
2.5	A example of a stochastic CG lexicon . . . . .	37
2.6	The possible lexical entry sequences of the example sentence . . . . .	38
2.7	An example of a probabilistic parse in stochastic CG . . . . .	39
2.8	An example P-CFG . . . . .	41
2.9	The CFG parse of the example . . . . .	41
2.10	A similar stochastic CG lexicon . . . . .	41
2.11	The CG parse of the example . . . . .	41
2.12	The CG parse of the example . . . . .	42
2.13	The CFG parse of the example . . . . .	42
3.1	The computational model for learning syntax . . . . .	45
3.2	Some possible parses when the closed-class word categories are set . . . . .	51
4.1	A graphical representation of the Transformation-Based Error-Driven Learning model . . . . .	68
4.2	An example of a Markov Chain . . . . .	71
4.3	An example tri-gram model . . . . .	72
4.4	An example Of a Hidden Markov model . . . . .	73
4.5	The HMM training algorithm . . . . .	77
4.6	An example of substitution in STSG . . . . .	89
5.1	A diagrammatic representation of CLL . . . . .	97
5.2	Some sample unannotated examples provided to CLL . . . . .	99
5.3	Some sample annotated examples provided to CLL . . . . .	100
5.4	Possible parses for examples . . . . .	105
5.5	An example lexicon . . . . .	108
5.6	An example of a lexicon that could be sent to the parser . . . . .	108
5.7	Collins [44] pseudo-code algorithm for a CKY parser for PCFGs . . . . .	110
5.8	The modified CKY parsing algorithm used by CLL . . . . .	112
5.9	A diagrammatic representation of the parse selection module . . . . .	114



5.10	Initial analyses of the examples that have already been processed . . . . .	120
5.11	The current state of the lexicon in the worked example . . . . .	120
5.12	The hypothesised parses for the next example . . . . .	121
5.13	Hypothesised lexicon 1 resulting from parse 1 . . . . .	123
5.14	Hypothesised lexicon 2 resulting from parse 2 . . . . .	123
5.15	The changed parse when reparsing with the first hypothesised lexicon . . . .	123
5.16	Modified lexicon 1 . . . . .	123
6.1	An example showing the incrementality of top-down parsing . . . . .	128
6.2	An example showing the lack of incrementality of bottom-up parsing . . . . .	128
8.1	The CF-PSG used to generate GC1 with example lexical entries . . . . .	142
8.2	Examples from GC1 . . . . .	142
8.3	The extra rules required for generating GC2 with example lexical entries . . .	142
8.4	Examples from GC2 . . . . .	142
8.5	Examples from the LLL corpus . . . . .	143
8.6	An example from the Penn Treebank with structural and part-of-speech an- notation . . . . .	145
8.7	The structure of the system for extracting corpora from the Penn Treebank .	147
8.8	Another example from the Penn Treebank with structural and part-of-speech marking . . . . .	148
8.9	The example after the translation module is applied . . . . .	148
8.10	The example with possessives combined . . . . .	148
8.11	The example with punctuation removed . . . . .	149
8.12	The example with nominals joined . . . . .	149
8.13	The two possible versions of the example written to the sentence corpora . .	150
10.1	Examples of entries in the lexicon from the output of the learning system . .	161
10.2	Examples of parses from the output of the learning system . . . . .	161
10.3	Branching examples: (a) right branching, (b) left branching, (c) random branching . . . . .	164
10.4	A tree with constituents marked . . . . .	168
10.5	An example with categories assigned . . . . .	169
10.6	Example of the output of Stage 1 . . . . .	170
10.7	Example of the output of Stage 2 . . . . .	170
10.8	Example of the output of Stage 3 . . . . .	170
10.9	Example of the output of Stage 4 . . . . .	171
11.1	A graph showing the increase in example size through the corpora . . . . .	190
11.2	A graph showing the lexicon growth for experiment 1 . . . . .	191
11.3	A graph showing the change in average crossing-bracket rate for experiment 1	192
11.4	A graph showing the change in lexical accuracy for experiment 1 . . . . .	192

# Acknowledgements

While completing the work on my thesis, I have become very aware of the impact of other people on me throughout the work and the debt of gratitude I owe them.

Firstly, the person who has, without doubt, had the largest impact upon the content of this thesis is my supervisor, Suresh Manandhar. Throughout the entire time he has been my supervisor we have met regularly to discuss my work and he has been a source of a great many ideas in that time and has continually encouraged me to keep going.

I must, of course, also thank my family, especially my parents, who have been a great support for my entire life. They have always encouraged me to use my mind and have given fine examples of how it should be done. Their kindness and encouragement throughout my research and their very literal support during my final year has been much appreciated. Special thanks must go to my father, who spent a large amount of time reading through sections of the thesis and improving my grammar and my writing style.

Thanks should also go to the AI group in the Department of Computer Science at the University of York. It is members of that group (past and present) who originally stimulated my interest in the fields of Artificial Intelligence and Natural Language Processing and I have been grateful for many useful discussions. Special thanks should go to Simon Anthony, a past member of the group and a close friend. He is another who spent significant time reading parts of my thesis and improving them. His impact on my academic growth is beyond measure and I am very grateful to know him. Thanks should also go to Alistair Willis, whose insight into Natural Language Processing often brought me up short and caused me to think significantly more clearly. I am also grateful to Jonathan Pickering who kindly provided me with the L<sup>A</sup>T<sub>E</sub>X class I have used for the thesis.

I am very thankful for the many friends, who have made the enjoyable times all the better and the less enjoyable times more bearable. Thanks goes especially to the friend who brought me back to York and completely changed my life. I doubt I was worthy.

Most fundamentally, I thank God. I thank him for the provision of all the above and for forgiving me and giving me life to the full, neither of which I deserve.

“When my heart was grieved and my spirit embittered, I was senseless and ignorant; I was brute beast before you. Yet I am always with you; you hold me by my right hand. You guide me with your counsel and afterwards you will take me into glory. Whom have I in heaven but you? And earth has nothing I desire besides you. My flesh and my heart may fail, but God is the strength of my heart and my portion for ever.”



# Declaration

This thesis has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree other than Doctor of Philosophy of the University of York. This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by explicit references.

Papers I have co-authored which pertain to this thesis are listed below.

Stephen Watkinson and Suresh Manandhar. Unsupervised lexical learning with Categorical Grammars. In Andrew Kehler and Andreas Stolcke, editors, *Proceedings of the Workshop on Unsupervised Learning in Natural Language Processing*, pages 59–66, 1999.

Stephen Watkinson and Suresh Manandhar. Unsupervised lexical learning with Categorical Grammars using the LLL corpus. In *Proceedings of the Workshop on Learning Language in Logic Workshop (LLL’99)*, 1999.

Stephen Watkinson and Suresh Manandhar. Unsupervised lexical learning with Categorical Grammars using the LLL corpus. In James Cussens and Sašo Džeroski, editors, *Learning Language in Logic*, volume 1925 of *Lecture Notes in Artificial Intelligence*. Springer, 2000. Expanded from the paper above.

Stephen Watkinson and Suresh Manandhar. A psychologically plausible and computationally effective approach to learning syntax. In Daelemans and Zajac [51], pages 160 – 167.

Stephen Watkinson and Suresh Manandhar. Translating treebank annotation for evaluation. In *Proceedings of the Workshop on Evaluation Methodologies for Language and Dialogue Systems, ACL/EACL 2001*, 2001.

Stephen Watkinson and Suresh Manandhar. Acquisition of large scale Categorical Grammar lexicons. In *Proceedings of the Meeting of the Pacific Association for Computational Linguistics (PACLING 2001)*, 2001.

I hereby give consent for my thesis, if accepted, to be made available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

# Chapter 1

## Introduction

Artificial Intelligence (AI), using Minsky’s [91] much-quoted [20, 45] definition

“is the science of making machines do things that would require intelligence if done by men.”

This has, of course, raised the questions of what exactly intelligence is and which tasks show it. Although no generally accepted answers have been produced to these questions, research in AI has continued to grow since the early work of the 1950s and the early conferences that attempted to define the field at Dartmouth, which are discussed by Crevier [45].

Instead of pursuing an exact definition of intelligence, most researchers have adopted a more pragmatic approach and attempted to get machines to solve problems which have intuitively seemed to require intelligence, in particular tasks that seem to require human intelligence. Perhaps two of the most important areas that have emerged have been Natural (i.e. human) Language Processing (NLP) and Machine Learning (ML).

NLP is the study of computer processing of human language both in understanding human utterances and in generating understandable and coherent utterances. While it was initially hoped that the use of computers for NLP tasks would be reasonably simple (see for example the descriptions in [45, 57]), it has become clear that this is a vast and difficult problem. However, it has become one of the key areas of AI study.

ML is the study of approaches to getting computers to learn concepts. Simon [112] defines learning as follows.

“Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and effectively the next time.”

The aim, therefore, is to provide machines with the ability to acquire and reason with information to perform better on some task or tasks. While this is obviously *useful* in the context of many systems (human or computer), it is also thought to be *intelligent*. Michalski *et al* [89] for example, state that

“The ability to learn is one of the most fundamental attributes of intelligent behaviour.”



While both these areas are of great importance to the AI enterprise, it has become clear that they need to be studied together. It is important to investigate how computers may learn natural language. Humans all learn to handle all kinds of language tasks both for understanding and generating spoken and written utterances. In this context, it is important when building NLP systems that the appropriate components, e.g. the grammar or the meaning of words, can be learnt. If these components cannot be learnt, then the NLP system is very unlikely to represent human knowledge of language accurately, or to deal with language in a way that is going to have any relation to the way a human would. Such a system would be likely to have severe limitations when attempting to interact with Natural Language.

The work presented here is an investigation of part of the task of getting computers to learn human language, thus combining both the areas of NLP and ML. In particular, the problem of acquiring syntactic information – especially the lexicon (the mapping between words and their syntactic roles) – is explored. The aim is to investigate ways of getting machines to learn this syntactic information in a way that is useful from an engineering point of view. It will be seen that this also bears some relation (perhaps not surprisingly, given that the issues within NLP cannot be dissociated from the human use of language) to the environment and methods used by humans in learning language.

## 1.1 General Issues

There are a number of general issues that raise themselves when developing NLP systems. Some of the most fundamental are discussed here as they will be fundamental to the rest of the discussion in this chapter and, in fact, to the rest of this thesis.

Perhaps the most important of the issues considered when developing the syntax learner, CLL, was that the system could be practically useful. By this, it is meant that engineering considerations were very important. The learning model should be defined such that it addresses solving useful problems, i.e. problems that really exist and that there is value in solving. The model should also address providing useful solutions, i.e. the output from a system, should aim to be, of itself, useful. This issue is discussed in more detail in Chapter 3.

The second issue considered is with respect to the type of Artificial Intelligence system that I want to build. Inherently, the perspective of the research presented here is that of symbolic AI. It could be argued that language and particularly syntax are fundamentally symbolic. However, in recent times, a huge amount of work in NLP has focussed on using non-symbolic methods, in particular statistical methods (see discussion in Chapters 3 and 4). The advantages of both symbolic and statistical methods are discussed in more detail in Chapter 3. However, it has become clear that statistical methods have a significant amount to offer with respect to NLP systems. The aim in this work has been to take some, essentially symbolic, methods and augment them with simple and efficient statistical methods. Hence, one of the issues that is investigated in the thesis, is whether these simple statistical methods can be integrated usefully with symbolic techniques.

The third issue is, to some extent, a result of the first. Seeking to solve useful problems with respect to natural language tends to lead one into solving human-like problems, as the

types of real problem that exist often bear a strong relation to the types of real problem that humans have to solve themselves. Hence, it will be interesting to compare the type of problems being solved with those solved by humans. It also follows that it may be interesting to consider how humans may solve these problems and how these human approaches may be useful in informing the computational approaches used. Chapter 7 summarises the pertinent psycholinguistic work.

## 1.2 Why Learn Natural-Language Syntax?

In the context of the issues described, it is now appropriate to consider why it is interesting to build systems to learn natural-language syntax. In the early days of AI it was assumed that enabling machines to deal with language would be relatively easy [45]. In fact, it has turned out to be both a huge and complex task containing a variety of sub-tasks such as speech recognition, parsing, semantic processing, and anaphora resolution. The complexity has been shown by the wide variety of studies in linguistics and computational linguistics that have been carried out in the past one hundred years in particular.

From a linguistics perspective, syntax is a vital building block upon which the majority of both computational NLP systems and theoretical studies of language will need to be based. The syntax of a language determines the structure of the language, or more precisely what structures are allowed and so which utterances are part of a language and which are not. The syntax can, in some senses, be considered to be mostly separate from the semantics, i.e. the meaning of those utterances. It is simply the definition of what makes up the language rather than what makes sense to us as humans. Following this view, the restrictions upon the meaning of an utterance are dealt with under semantics (another section of linguistic theory).

This separation provides an indication of the way linguists have investigated language, which has been to break the area down into smaller sub-areas, e.g. syntax and semantics. While there is no doubt that these areas interact, they form areas that it is helpful to study separately.

However, a number of NLP systems do not have syntactic components and go straight to a semantic representation, or simply use stochastic methods to extract the relevant information, e.g. [122, 121, 123, 147, 64] (see also Allen [4] for a discussion). If this is possible, then why not ignore the syntactic component? Although it is possible to achieve a great deal by going straight to the semantic representation, for a full natural-language understanding or generation system it is generally considered necessary to include the full complement of syntactic knowledge to be able to build sentence and discourse meaning accurately and so it is very valuable to be able to explicitly learn syntax.

Unlike the syntax of formally defined languages, such as programming languages, the syntax of natural language is exceedingly difficult to define. This is due to the fact that all syntactic rules, or generalisations, must be made on the basis of observed data. This process is the opposite of the process involved in defining formal languages, where the data is the result of the syntax rules, which have been pre-defined. Not only is it difficult to define grammars for natural languages, it is also difficult, from a computational perspec-



tive, to determine how best to represent and manipulate syntactic information (this issue is discussed at length in Chapter 2). While significant efforts have been made with respect to hand-building NLP resources, such as grammars and annotated corpora, it has proved to be expensive and to return incomplete, noisy and brittle results (see some discussion in Chapters 3 and 4).

Hence, we have a useful problem that, if solved, could return useful results. If it were possible to effectively learn and modify grammars, then the expensive and ultimately flawed hand-building approach could be avoided and potentially better results could be achieved. These results – grammars – are then very useful with respect to building very many useful NLP systems. Hence, the general syntax-learning problem is well motivated from the engineering perspective.

The syntax learning problem also lends itself to the extension of symbolic methods with simple statistical methods. Syntax, by very nature, is symbolic. The aim is to capture knowledge representing the structure of sentences of natural language. Both the sentences and the structure are symbolic. However, statistics can be useful in building more robust grammars and in building grammars that can rate different parses for sentences determining the best, or set of best, analyses for a given sentence and grammar. While the task is still essentially symbolic, simple statistical methods may be enough to perform these tasks well enough for many NLP tasks. Chapter 2 discusses both the symbolic grammar formalisms and their statistical extensions. There is also a further use of statistical methods. As well as rating particular analyses of sentences, they can also be used to rate different options in the learning process, e.g. which grammar to select next in the search space. This is discussed further in the next section.

Finally, the syntax learning problem is interesting from a psychological perspective. As mentioned above, in many cases in NLP, solving the practical problems of building NLP systems leads to solving problems similar to those faced by humans. Syntax learning clearly occurs in humans, and, as will be discussed in Chapter 7, there is much that can be learned both with respect to the human learning environment and human learning methods.

### 1.3 The Syntactic Model

Before defining any learning model or algorithm, it is useful to consider what exactly is to be learned. While I have already said that the aim is to learn syntax, it is necessary to determine what kind of syntax, how it is represented and what parts of the syntax are already known.

Fortunately, there has been extensive study of syntax, both in the way language actually works and the ways it can be formally represented. This provides us with three very useful products:

- an approximate goal,
- a way of representing syntactic knowledge,
- a set of constraints on the search space.

Firstly, it is vital when attempting to build a learning system that it can be evaluated. The best way to evaluate a learner is to know what it was supposed to learn and then, in some way, to deduce how close the learner is to having learnt it. Linguistics has provided us with a set of syntactic principles which can be considered to define the kind of knowledge that the system will be used to induce. Computational linguistics provides us with resources, like the Penn Treebank [88, 85], which apply this knowledge to a large corpus of naturally occurring language. Together these provide a “gold standard” of the type of syntactic information that needs to be learned and the way it needs to be applied, i.e. the goal of a system. I discuss this syntactic information in Chapters 2 and 8. I also discuss the detail of an approach to translating the annotation of the Penn Treebank into alternative annotations to make it a more useful “gold standard” in Chapter 10.

Secondly, research in linguistics and computational linguistics provides a variety of formalisms in which to represent the syntactic knowledge that has been discovered. These formalisms are discussed in detail in Chapter 2, along with a variety of ways of using stochastic information to enrich purely symbolic formalisms. Given the recent interest in stochastic formalisms in computational linguistics and the value of combining symbolic and stochastic methods, I have developed and used a simple stochastic version of Categorical Grammar (CG) [118, 144]. The stochastic model is the same as that of Osborne and Briscoe [96] and is essentially a uni-gram model.

CG was selected as the formalism for three major reasons. Firstly, it is lexicalised, which means that, where there were two learning problems – learning a lexicon and learning a grammar – there is now only the problem of learning the lexicon, which is the grammar. Secondly, the grammar can be extended easily to use statistical information. Thirdly, the elegant, structured nature of the categories (in which the majority of the grammar is contained) allows for future extension of the work, e.g. the automatic generation of categories.

As yet, the CG formalism has not been explored as far as would be desired. The system reported here does not have mechanisms for handling all syntactic phenomena (e.g. movement). In future, the grammatical model will need to be extended to allow broader coverage.

If it is assumed that a certain amount of the CG formalism is available to the learner (see the next section), then grammar-learning becomes a matter of learning a large scale lexicon for CG, where the lexicon contains the majority of the syntactic information. Hence, the implementation of the learner is a Categorical Lexicon Learner (CLL), building a probabilistic mapping between words and the syntactic categories that they can take.

## 1.4 The Machine-Learning Model

There are a large number of ways of defining the computational setting of a natural-language syntax-learning system. In this section, I discuss the type of machine-learning problem that is being addressed in this work. This can be summarised with respect to four issues:

1. the type of examples the learner receives;
2. the other external evidence that is available;



3. the kind of background knowledge that is available to the learner and
4. the type of learning mechanisms that are used.

Each of these is discussed in turn below.

There are two main issues to consider when determining the type of *example* a learner receives. Firstly, the examples may or may not be annotated, i.e. the learning model may be *supervised* or *unsupervised*. In fact, there is a continuous nature to this distinction, as there are different degrees of supervision, i.e. different amounts of annotation. In the model defined, both a completely unsupervised setting, where the examples are simply lists of words, and a weakly supervised setting, where the examples have all the nouns and verbs annotated (with N and V respectively) as such are considered.

Supervised methods are avoided for good practical reasons. Getting correctly annotated data is very difficult and potentially very expensive. While there are annotated corpora available (see Chapter 8 for a discussion of some) there are a number of problems. Corpora are not available for all languages or domains. It is also difficult to obtain corpora that are annotated with the grammatical formalism in which one is interested. If a suitable corpus is not available, then it is difficult to build one that is suitable. It either involves difficult and imprecise automatic annotation (see Chapter 10 for a summary of some work in this area), or people must be paid to annotate a sufficient quantity of text.

In contrast, unsupervised methods can use any text that is available. In these days of huge amounts of text being available in computer-readable form, this means that there are large amounts of text available for most domains and most languages. If unsupervised methods for learning syntax can be shown to be effective, then it should be possible to learn grammars for all kinds of different NLP systems at a minimal cost. As Kehler and Stolcke [75] suggest, unsupervised learning methods appear to be the future of learning for NLP tasks.

As an aside, it would appear that humans learn from unannotated examples. This is discussed in more detail in Chapter 7, for the moment, suffice it to say that, as would appear intuitive, the spoken language from which humans learn does not seem to have any explicit annotation (although the context of the utterance may imply some annotation).

Unsupervised learning problems are, however, much more difficult. Commonly, it is necessary to use more powerful learning techniques and more background knowledge for unsupervised learning. However, as will be discussed, it is often cheaper in terms of time and human effort to provide these techniques and this knowledge than it is to build the large annotated corpora required for supervised learning.

The second issue with respect to the type of examples which the learner is to be supplied with, is whether or not the system receives negative examples, as well as positive examples. Positive examples are simply examples of correctly formed natural language utterances (assuming that they have no annotation). It is less clear, in the unsupervised setting, what negative examples comprise of. Essentially, they are examples which are syntactically incorrectly formed with respect to the language. Quite what sort of errors are interesting or useful is open to question, as is how such negative examples can or should be generated. Perhaps the most important issue is that, if both positive and negative examples are to be used, then

they must be marked as such (a kind of weak supervision). This is in contrast, for example, with a child learning and hearing a badly formed sentence, which is noise rather than a negative example, as the child is unaware of the fact that the sentence is badly formed.

The learning model presented here is positive-only, i.e. only positive examples are supplied. This is practical for two main reasons. Firstly, lots of positive examples are available, i.e. there are lots of examples of correctly formed computer-readable natural-language text available. Secondly, there are not lots of negative examples available, for the reasons described above, i.e. it is hard to determine what type of negative examples are required and it is hard to generate those examples.

Again, the positive-only learning model can be viewed as being psychologically plausible, as indicated above. Children learning language do not seem to receive explicitly marked negative examples.

While negative examples may not be available, it is still possible to provide a learner with negative *evidence*, i.e. evidence about grammatical incorrectness. For example, a syntax-learning system could hypothesise a grammar, generate some examples and use an oracle to determine whether the examples were part of the language (Adriaans [2] uses such an approach). When the oracle rejects ungrammatical examples, it is providing negative evidence. This can be seen as modelling the correction parents give to children (although whether or not this is correct is discussed in Chapter 7). This method of learning, *query-based* learning, is discussed in Chapter 3. It was decided not to provide the system with negative evidence. From an engineering perspective, it is hard to determine efficient ways of providing negative evidence. For example it is not clear how to build an oracle with broad coverage for the syntax of natural language, as in many cases the syntactic knowledge will not be available. Hence, the system learns from positive evidence (the examples), but does not receive any negative evidence.

It is also important to consider what *background knowledge* is available in the learning model. Firstly, a set of syntactic rules for CG are provided. As CG is a lexicalised formalism, this is a small set (either two or three rules).

Secondly, a set of CG categories is provided. This is a substantial amount of syntactic information, as, due to the lexicalised nature of CG, these are complex categories, which specify modifier and argument relationships. This set is intended to be all the categories that the grammar requires and is unlikely to contain categories that are not used. As such this set is language specific. Given the strong nature of this background knowledge, it is hoped that in the future the set of categories can be replaced by a set of category schema, or a mechanism for generating new categories when needed. However, this is still less knowledge than a supervised algorithm where the words are labelled with categories, as the mapping between words and categories and the context of particular instances are also included in this annotated data.

The final sort of background knowledge is an initial, bootstrapping, lexicon. This is not used in all experiments and different sizes of lexicon are used in different experiments. The lexicons used consist of a mapping between closed-class words and the CG categories. These lexical entries can be used with the appropriate word in an example to restrict the categories that other words can take, i.e. they constrain the search space of possible category



assignments for words. The number of closed-class words in sentences means these constraints are useful, but do not make the problem trivial.

The background knowledge is discussed in more detail in Chapters 2, 3 and 5. It should be noted at this point that all these knowledge sources are easy to build – much easier to build than an annotated corpus for example. Hence, the engineering motivation is maintained. For example, a system using this model could easily be ported to another problem by re-building the background knowledge, whereas a supervised system would not allow this. However, it is clear that there is a trade-off occurring here. By insisting upon using an unsupervised model, much more background knowledge is required.

It should be noted that the background knowledge can, to a limited extent, be motivated by psycholinguistic research. It would appear that some lexical information is available to children when they are acquiring syntax and that they have some innate syntactic knowledge. However, the extent of the knowledge provided here, in particular the language-specific elements, is not psychologically plausible.

None of the previous parameters have really dealt with the actual *learning mechanisms* to be employed. Some principle(s) needs to be used to guide the system towards appropriate hypotheses and to determine what is and what is not a good hypothesis.

The learning model uses one essentially symbolic mechanism. Each example is parsed in turn (the system is somewhat incremental in this way). The current lexicon (that is what has been learned so far along with any initial lexicon) is used for this parsing process. In this way, the syntactic constraints provided by the CG categories are applied to reduce the possibilities for categories for unknown words. Parsing is then applied to previously seen examples to determine changes that could be made to previous parses and so modifications to the lexicon that is learned. This reparsing approach was settled upon due to early empirical evidence, which suggested that without reparsing the learner was not able to correct early mistakes. However, the efficiency costs of this reparsing, even if it is done intelligently, suggest that some other approach should be investigated in the future. The storage of the examples that have been seen is an area where CLL is not psychologically plausible.

The learning model also uses two principles, which are essentially statistical at heart. The first is the Maximum-Likelihood (ML) principle, where an algorithm, on the basis of a probabilistic model, attempts to return the most likely, or a set of the most likely, solutions. The learning model uses the ML principle for determining possible syntactic annotation for sentences. Such an approach has been well-used and motivated in the past (see discussion in Chapters 2 and 4).

The second machine-learning principle used is compression, where, the smallest, or most compressive result, is considered to be the best. Two simple and efficient methods are investigated (in line with the aim to investigate simple statistical additions). Compression has been used for syntax-learning before (see the discussion in Chapter 3) and allows efficient and potentially even psychologically plausible learning.

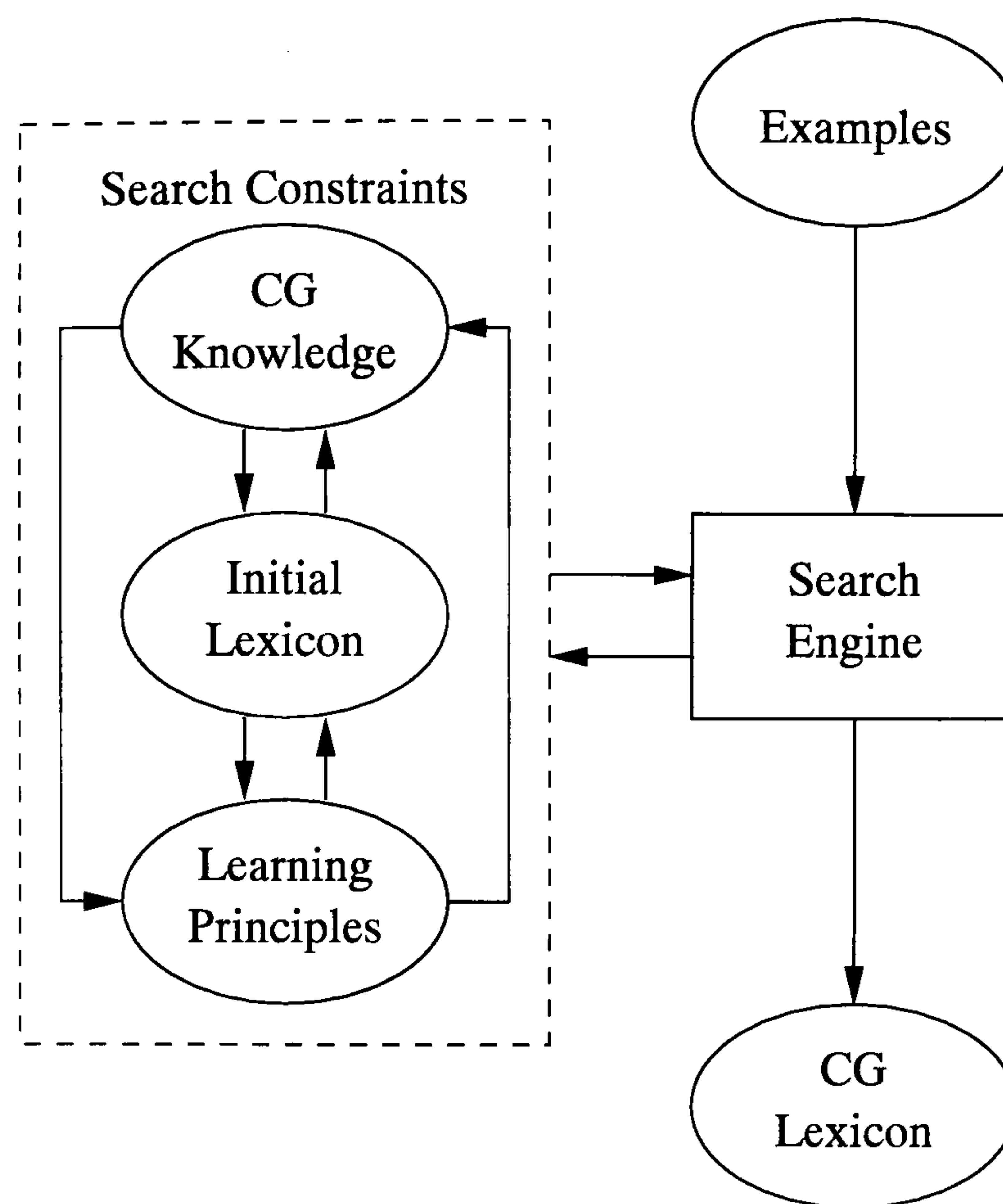


Figure 1.1: The computational learning model

## 1.5 A Summary of the Learning Model

Figure 1.1 provides a diagrammatic view of the learning model. The system receives unannotated positive examples. A search engine (a combination of a parser and a method of using the parser to calculate the most compressive lexicons) that investigates the most plausible syntactic annotations for the examples is provided. Those annotations that are selected are used to build Categorical Grammar lexicons. This model is used to build a (probabilistic) mapping between the words in the examples and the CG categories provided in the background knowledge.

As with most AI problems, the search constraints will be critical in determining the success or failure of the system. In this model, search is constrained by syntactic information in the form of the CG rules and categories; the initial bootstrapping lexicons and the learning principles, i.e. compression and ML.

## 1.6 Investigating the Model

CLL, an implementation of the model has been built. The approach is to apply a parser to each example in turn and build up a lexicon derived from the best parses. The definition of best comes partly from a probabilistic parsing process, which generates the  $n$ -best parses using the current lexicon. The best parse is then chosen from this set of  $n$  by looking for the one that leads to the most compressive lexicon (it is this process that requires the reparsing of previous examples). Chapter 5 describes this model in detail.

CLL has been tested on a number of corpora (described in Chapter 8), including a large section of the Penn Treebank, although not including null elements, i.e. movement. Hence



a large range of examples of the syntactic constructions of English have been given to the learner. The system has been applied to these corpora using a variety of parameter settings (e.g. compression metric, size of  $n$  for the parser, initial lexicons). Chapter 9 discusses the experiments in detail. Currently the largest corpora upon which the system has been used can only be considered to be relatively small (all the corpora are less than 100,000 words). This is essentially due to time and efficiency constraints and is discussed further in Chapters 5 and 8.

The results are encouraging if not, as yet, entirely conclusive, especially when compared with the results of other systems on simpler (e.g. supervised) problems. Unfortunately, there appears to be no directly comparable system against which to measure success, due to the fact that other systems solve somewhat different problems (e.g. supervised, or disambiguation problems). However, very high accuracy results on simple corpora, followed by a crossing bracket rate of around 4 on a corpus extracted from the Penn Treebank is encouraging as a first step. Methods of evaluating the results are discussed in Chapter 10, including a large section on translating the annotation of the Penn Treebank from the phrase-structure annotation to a CG annotation to allow for labelling-accuracy measures. The results are presented in detail in Chapter 11.

Options for further investigation open up in many directions; among these are:

- inclusion of movement, so entire NL corpora can be used;
- development of a more complex stochastic grammatical model to allow context to disambiguate;
- gradual removal of the less plausible linguistic knowledge given to CLL and replacement with learning techniques;
- development of the translation of the Penn Treebank annotation such that it provides a translation that is closer still to our needs;
- a broader investigation of the possible parameters, e.g. using alternative definitions of compression;
- developing the CG to include more syntactic information, e.g. the ability to distinguish tense, number and person;
- investigating the possibilities of cheaper parsing;
- extending the system to work on utterances (even spoken language corpora) rather than sentences.

A large number of these extensions will require serious consideration of efficiency, particularly with respect to time, and may require a large degree of re-implementation.

## 1.7 Questions Addressed in the Thesis

The following list summarises the questions addressed in this work.

- Does the learning environment described above, i.e. unsupervised and positive-only, allow a useful syntax-learning system to be built?
- How useful is it to extend symbolic language models with simple statistical models when investigating the natural-language syntax-learning problem?
- What kind of background knowledge is most useful in unsupervised learning using our learning model?
- Are the compression and maximum-likelihood learning principles appropriate for this learning task?
- Do the compression and maximum-likelihood learning principles combine effectively with each other and the symbolic methods in this learning model?
- Does the lexicalised nature of CG help in the learning of syntax?

This is a large range of questions, which will be only partially answered by the work presented here, however, each question has been addressed and some insights and conclusions have been achieved with respect to each.

## 1.8 The Contributions

The major contributions of the work presented here are listed below.

- An unsupervised, positive-only learning system has been built and tested on a range of corpora, including restricted sub-corpora extracted from the Penn Treebank.
- A weakly supervised system (a small modification of the unsupervised system) has similarly been built and tested.
- The parameters of both these systems have been investigated allowing some conclusions to be drawn with respect to the best settings of the systems.

There are also some less significant contributions of the work, that have arisen largely as a by-product of the major contributions.

- For evaluation, it was necessary to build some CG annotated corpora, to achieve this, a preliminary investigation of a method for automatically translating the Penn Treebank annotation into a CG annotation has been investigated.
- Both the learning system and the treebank translation system have allowed the production of a number of CG lexicons and CG annotated corpora. The accuracy of these lexicons and corpora is currently quite low, but they may be useful as a starting point for other research.
- A relatively efficient  $n$ -best probabilistic parser for stochastic CG grammar has been built for use within the learner.



## 1.9 Conclusion

In conclusion, this DPhil thesis presents a large variety of work that is of interest to the NLP and ML communities and particularly of interest to the Natural Language Learning community. The aim has been to investigate approaches to building unsupervised learning systems for learning Categorical Grammar lexicons. It is suggested that such systems would be solving a useful problem and could return useful results.

The work has raised a large number of interesting and important issues, which have been investigated to some extent, although there is clearly broad scope for further work. However, the results presented here are encouraging and indicate that this kind of work can fruitfully be pursued.

The remainder of the thesis has been referenced frequently above. However, the overall structure is presented in detail below.

**Chapter 2** In this chapter I discuss a variety of possible grammatical formalisms that could be learned and the type of linguistic constraints that can be applied to the learning problems using these formalisms. A stochastic extension of CG emerges as the most appropriate formalism.

**Chapter 3** This chapter describes the computational learning model that has been developed and discusses some of the computational constraints that need to be considered.

**Chapter 4** A large number of other systems have attempted related problems and have influenced the work presented here. In this chapter they are reviewed and compared with CLL.

**Chapter 5** The implemented instantiation of the computational learning model, CLL, is described in detail in this chapter.

**Chapter 6** The issue of parsing within CLL is discussed in detail in this chapter, with particular attention being paid to efficiency and psychological plausibility.

**Chapter 7** This chapter describes the implications of psychological and psycholinguistic work on both the definition of the syntax-learning problem and the type of techniques that may be involved in solving it.

**Chapter 8** The corpora with which the system is trained and tested are described in this chapter, including a description of their construction where appropriate.

**Chapter 9** A large variety of experiments have been carried out. This chapter describes them and presents the rationale behind how CLL has been tested.

**Chapter 10** Evaluating an unsupervised learning algorithm is not always trivial – especially when using a less common syntactic formalism like CG. This chapter presents a variety of appropriate techniques for evaluation, including a significant project in translating the annotation of corpora such as the Penn Treebank.

**Chapter 11** The results that have been produced by the experiments are presented and evaluated in this chapter.



**Chapter 12** The final chapter of the thesis discusses the successes and weaknesses of the work. A number of other areas of work that should follow on from this work are suggested and motivated as well.

**Appendix A** The Penn Treebank part-of-speech tag sets are given in this appendix for reference.

**Appendix B** The initial lexicons given to the learner for boot-strapping in the experiments are given here.

**Appendix C** This appendix contains the categories databases with which the learner was supplied for the experiments.

## Chapter 2

# Grammatical Formalisms

A system for learning natural language requires a method for representing the grammatical information. There are many formalisms (i.e. notations for describing the grammatical information) from which to choose, so a list of requirements has been described in Section 2.1 to enable evaluation of the different options. These requirements are based in part on the discussions of the previous chapter and in part on further linguistic and computational requirements.

There are far too many grammatical formalisms for each to be considered in turn, so in Section 2.2, they are considered with respect to some of the features upon which they may differ. A formalism may be lexicalised, stochastic and unification-based. The formalism itself is not the grammar, rather the grammar can be represented using the formalism, giving lexicalised, stochastic and unification grammars. Each of these features is discussed in turn and for the current work a stochastic lexicalised grammar is settled upon, without having any unification component.

While there are still a number of valid possibilities, Categorical Grammar (CG) is selected as a formalism that fits the requirements well. In Section 2.3, the CG used is described in detail along with the stochastic model with which it is extended.

### 2.1 Requirements

A grammatical formalism for the learning system described in Chapter 7, which must be linguistically appropriate and computationally practical will have a number of requirements, some of which must be met and others which will be highly desirable. In this section, the requirements of the grammatical formalism are described in detail.

#### 2.1.1 Expressiveness

A grammatical formalism for representing the grammar of a natural language must be adequate for the task, i.e. it must be expressive enough to describe which structures are to be allowed in the grammar and which are not. However, it is important from a computational perspective that the formalism is not too powerful and thus too computationally expensive to use effectively. English, the language upon which the learner is to be used, is generally considered to be mostly context-free with respect to the Chomsky Hierarchy [21, 4]

(although there is some discussion about the need for context-sensitive structures in certain circumstances [21, 57]). For a good discussion of the Chomsky Hierarchy see for example Aho and Ullman [3]. Fortunately, there are various computationally effective algorithms for context-free grammars, in particular, parsing algorithms for building syntactic analyses (see Chapter 6). Hence, the ideal formalism would be one with the expressive power equivalent to a context-free grammar, with the potential for extension to a mildly context-sensitive grammar. Preferably, it would also be possible to use or modify already existing algorithms to use the formalism.

### 2.1.2 Learning Problem Issues

Ideally, the formalism should allow the learning problem to be stated and solved simply and elegantly. In other words, the formalism should not make the learning problem more complex.

### 2.1.3 Extensibility

The formalism should be easily extensible either to include further syntactic structures (e.g. for other languages) or to allow the inclusion of other non-syntactic information. This information could be part of what is to be learned, or it could be used by the learner in the learning process, or used with the learned syntactic information after learning has occurred. In particular, it is important to use a syntactic formalism that relates well to semantic information that needs to be used in most NLP systems.

### 2.1.4 Separation of Knowledge

It is important to be able to easily separate the types of knowledge that will be used in the system. There are two major types within the learning system.

- background knowledge
- knowledge to be learned

Ideally, the system can easily be supplied with appropriate background knowledge, which is then used to learn the remaining knowledge.

### 2.1.5 Elegance

Finally, from an aesthetic perspective, it would be desirable to express the linguistic knowledge elegantly. This has practical importance, because an elegant and clear description of the knowledge that is already possessed by the learner and that which is to be learned, keeps the current state of the syntactic knowledge clear. Similarly, an elegant description of the learned knowledge will allow better evaluation of the success, or otherwise, of the learning system.



## 2.2 Alternatives

In this section I discuss the different alternatives for grammatical formalisms with respect to three features with which I am concerned.

- lexicalisation
- stochastic extension
- unification mechanisms

These features are evaluated with respect to the requirements discussed above. It will become clear that different formalisms may have none, some or all of the features.

### 2.2.1 Lexicalised

A lexicalised formalism is one where most of the syntactic information in a grammar is stored in the lexicon, usually leaving just a few rules that define how lexical entries can be combined. Examples of such formalisms are Head-Driven Phrase-Structure Grammar (HPSG) [101], Categorical Grammar (CG) [144, 118] and Lexicalised Tree-Adjoining Grammar (LTAG) [72]. Lexicalised formalisms stand in contrast to more rule-based unlexicalised formalisms such as standard phrase-structure grammars (PSGs) [3], where lexical entries are simply rules that map a part-of-speech to the word; or Chomsky's Principles and Parameters theory [38, 47], where the lexicon is not generally handled in any explicit way.

In terms of *expressiveness* both lexicalised and unlexicalised grammar formalisms can be used for defining grammars with different positions in the Chomsky Hierarchy. In fact, different types of phrase-structure grammars are used to define the Chomsky hierarchy, by placing different restrictions upon the rules allowed in the grammar. Similarly, it is possible with lexicalised grammars to define them in ways that give them different levels of expressiveness. For example, standard (AB) CG [144, 118] (see Section 2.3) has been shown to be weakly equivalent to context-free phrase-structure grammar (i.e. for all context-free grammars, there is a CG that describes the same language and vice-versa) [11]. However, the CG formalism can be extended by adding extra rules for combining categories to allow the formalism to be context-sensitive or even unrestricted [118, 144].

For TAG and Combinatory Categorical Grammar (CCG), which is CG that has been extended by adding further combination rules [118, 119], it has been shown that their expressiveness can be beyond context-free, but significantly less than fully context-sensitive grammars [130, 131]. These grammars are known as *mildly context-sensitive* grammars and they can still be efficiently parsed (using the CKY algorithm – see Chapter 6) and can capture some of the appropriate further natural-language syntax that context-free grammars fail to capture [130, 131]. In future, it will be useful to investigate using these grammars so that further coverage can be achieved. However, for the sake of simplicity, it would seem wise to start with the less expressive formalisms.

With respect to the complexity of the *learning problem*, a lexicalised grammar appears to provide a more elegant definition of the problem. With a lexicalised formalism the learner has one task, that of building the lexicon. However, with an unlexicalised formalism, two

tasks must be completed: a large set of grammar rules or parameters needs to be learned and a lexicon must be built.

Lexicalised grammars commonly have an elegant link between syntactic and semantic analyses. The syntactic structure is defined to follow the semantic structure. Categorical Grammar is particularly strong in this respect, as the syntactic function-argument structure exactly mirrors the semantic function-argument structure. This has three advantages:

- the grammar will be suitable for semantic analysis.
- semantic information could be learned.
- semantic background knowledge can be used to aid learning.

Such a link is less easily achieved with unlexicalised formalisms, and so in this sense lexicalised grammars are better with respect to *extensibility*, as well as possibly containing useful features for aiding learning.

Both lexicalised and unlexicalised grammatical formalisms allow grammars to be built that are easily *extensible* with respect to syntactic phenomena. With unlexicalised formalisms it is possible to add further rules to incorporate new syntactic structures. It is perhaps more complicated with lexicalised grammars, where both lexical categories and category combination rules can be added to extend grammars (for example CCG discussed above).

Similarly, both lexicalised and unlexicalised grammars allow appropriate *separation of knowledge*. With a phrase-structure grammar some notion of known and unknown rules would have to be developed. Principles and Parameters theory [38, 47] has achieved this by parameterising the rules governing syntactic structure. Hence, the knowledge that is innate is represented by the rules or *principles* and the knowledge to be learned is the values of the *parameters*. This is perhaps the most elegant separation of knowledge in a formalism. An example of this is given in the  $\bar{X}$  theory (see Culicover [47] for more detail). This theory essentially describes a principle for the way phrases are composed. Example  $\bar{X}$  schema are:

$$XP \rightarrow XP \text{ Adj}$$

$$XP \rightarrow \text{Spec } X'$$

Here  $X$  can be considered to be a variable for all possible heads for phrases, e.g. verbs, nouns. *Adj* is short for adjunct and *Spec* is short for specifier. So for verbs one would have the rules:

$$VP \rightarrow VP \text{ Adverb}$$

$$VP \rightarrow \text{Spec } V'$$

where  $V'$  will specify the type of verb, i.e. intransitive, transitive, ditransitive. Importantly, in the schema, there is no ordering of the categories on the right-hand side. For example, the principle does not specify what order the specifier and verb must come in. This varies between languages and so cannot be assumed to be part of the universal grammar. Hence, it is instead thought to be a parameter set by the examples presented to the learner. So if the learner were to be presented with the sentence “John runs”, assuming some lexical knowledge, it would be able to make the inference that the order parameter be set to *Spec V'*.



However, within a lexicalised grammar it is also possible to provide a reasonable separation of knowledge. The most common approach would be to assume that the rules for combining lexical entries and the type of lexical entries allowed would be known, and the mapping between words and their lexical entries is the knowledge to be learned. The separation captured by the example above can also be captured by a lexicalised grammar. In Categorical Grammar for example, the categories, which are language dependent, express the position of the arguments and the rules, which are language independent, combine those rules. For example, for English, the category for a transitive verb could be  $(s \backslash np) / np$  (following Steedman's notation [118, 119, 144]), which states that the verb is a sentence (the  $s$ ) looking forward for a noun phrase object (the  $/np$ ) and then backwards for a noun phrase subject (the  $\backslash np$ ). Such categories can then be combined with the categories of other words using the Categorical Grammar rules.

In practice, lexicalised formalisms can be defined to be very *elegant*. It is clear where different types of knowledge reside, and they are designed such that most general syntactic principles, i.e. those for combining lexical entries and what lexical entries consist of, have been separated. However, it is to some extent a matter of taste, as to whether lexicalised or unlexicalised formalisms are preferred with respect to this criterion.

While it may be possible to use either a lexicalised or an unlexicalised formalism, I have decided to use a lexicalised formalism. Advantages noted with respect to the learning problem suggest that a lexicalised system will provide the more elegant learning approach and the greater possibility of extending the types of information provided to the learner and learned by the learner.

### 2.2.2 Stochastic Formalisms

A stochastic syntactic formalism is one which assigns a numerical likelihood to all possible analyses that can be generated for any given example. To be a stochastically well-defined the sum of all the numerical likelihoods of analyses for a particular sentence or phrase should be one. For example, suppose for the sentence:

“I hit the man with the bat”

a grammar only assigns two possible structures depending on whether the bat was used for hitting the man, or the the man who had the bat was the man who was hit. A well-defined probabilistic grammar would require that the probabilities of those two analyses summed to one.

Many stochastic grammar formalisms have been proposed. Generally these have involved extending well-known formalisms to include a probabilistic model, e.g. Probabilistic Context-Free Grammars (PCFGs) [33], Stochastic Categorical Grammar [95, 96] or Probabilistic L-TAG [72]. There have also been more specific attempts to build grammars that are defined specifically for stochastic uses, e.g. the Data-Oriented Parsing approach of Bod [18] or the formalism used by Collins [44].

In terms of *expressiveness*, the addition of stochastic information to a formalism does not, of itself, alter the expressiveness of a grammatical formalism when considered with respect to the Chomsky Hierarchy. However, the advantage of such a model is that it provides a

measure of the likelihood of analyses. In this sense, the expressiveness of the formalism is greatly enhanced. Such information can be of great benefit for disambiguating between different analyses. A learning system that can select the correct, i.e. most likely, analyses will be able to learn more effectively. Such an effect would be hard to implement with a non-stochastic approach.

Using a stochastic formalism has a large effect on the *learning problem*. In effect, another dimension is added to learning the grammar, as the appropriate likelihoods or probabilities must be learned. However, this can also improve the ability of algorithms to learn, as they have another dimension with which to control the search for possible grammars. This can be particularly useful in an unsupervised setting, as there may be relatively few options for controlling search. It should also be noted that humans appear to use frequency-based information as part of their learning processes (see Chapter 7) and thus it would seem an appropriate feature to use in this context.

Using stochastic formalisms can also allow for good *extensibility*, at least in the context of developing a more powerful and accurate model for disambiguation. Again this is not necessarily an extension of the coverage of the grammar, but rather an improvement of the model used to determine the correct analysis.

The use of a stochastic formalism makes very little difference with respect to issues relating to the *separation of knowledge*. In effect, it is assumed that it is known that likelihoods are attached to analyses. These likelihoods themselves are learned. This appears to be psychologically plausible if the likelihoods are related to frequency, as humans appear to use this kind of frequency information to learn (see Chapter 7).

With respect to *elegance*, the addition of a stochastic component can make a formalism somewhat fussy to use. However, it does allow a large number of difficult NLP problems to be handled in a particularly simple way. In general, it is probably true that the use of stochastic models will reduce the complexity of the systems needed to handle various problems and so can be considered to provide a simpler approach than a purely symbolic approach.

It seems wise, therefore, to use a stochastic model of grammar. Particular weight is given to the learning issues and expressiveness, as the addition of a stochastic component provides very useful information for narrowing the options down in an unsupervised setting, as will be seen when the system is described.

### 2.2.3 Unification Formalisms

A unification formalism is one that allows syntactic components to have variable arguments. The arguments are then instantiated using some form of unification. Commonly arguments are used to express features of a word or phrase, e.g. the form of a verb or the number of a noun. By using these arguments, it is possible to reduce the number of rules required. For example, consider the following phrase-structure rule for noun phrases.

$$NP \rightarrow Det N$$

This states that a noun phrase can be made up of a determiner and a noun. However, as it takes no account of number, this allows ungrammatical noun phrases, e.g. \*‘‘these man’’



and \*“a women”. To prevent this it would be necessary to have the following two rules, the first for singular determiners and nouns and the second for plural determiners and nouns.

$$\begin{aligned} NP &\rightarrow DetSing NSing \\ NP &\rightarrow DetPlu NPlu \end{aligned}$$

All lexical entries for determiners and nouns would need to indicate number and in some cases (e.g. “the”) multiple lexical entries would be required where the word can act in either way.

A unification grammar would simply add number as an argument and then require that the two variables in the rule unify, i.e. only the following rule is required.

$$NP \rightarrow Det(Num) N(Num)$$

In this case, words that can be either singular or plural do not require separate entries in the lexicon, but can simply be given a variable for the number argument.

A number of unification-based formalisms have been developed. The simplest are probably Definite-Clause Grammars (DCGs) [21, 4, 57]. The parameterised rule above can be considered an example of the type of rules that make up these grammars. They are simply context-free grammars that have been extended to allow parameterisation. More recently there have been more advanced unification grammars, for example Generalised Phrase-Structure Grammar (GPSG) [56] and Head-Driven Phrase-Structure Grammar (HPSG) [101], which use *feature structures* to express much of the syntactic information. A feature structure is essentially a set of *features*, where a feature can be anything related to the word or structure involved, e.g. person, number, complements, semantics, or orthography. HPSG is also a lexicalised grammar, so words are assigned feature structures that define the word and can include the complements it takes, possibly subcategorisation information, as well as the standard number and person information. Feature structures are usually represented using Attribute-Value Matrices (AVMs). Figure 2.1 shows a simple example of a feature structure (as an AVM) for a third person singular feminine verb (modified from a similar example in the work of Russell [104]).

$$\left[ \begin{array}{c} \text{INDEX} \left[ \begin{array}{cc} \text{PER} & 3 \\ \text{NUM} & \textit{sing} \\ \text{GEN} & F \end{array} \right] \end{array} \right]$$

Figure 2.1: An example feature structure

There have also been extensions of other formalisms to include the powerful benefits of unification, e.g. Categorical Unification Grammar (CUG) [124].

With respect to *expressiveness*, unification grammars can be exceedingly powerful, both formally and informally. Formally, it is simple to go beyond context-free and even context-

sensitive power depending on the type of rules used for application and the kind of features used (for example if any feature has an infinite number of possible values then the grammar is no longer context-free [21]). Informally, features allow the expression of many types of information, many of which will not be relevant. This may also lead to a problem with unification grammars, which is that it is possible simply to add features without any principle or constraint. Very large and inelegant grammars that capture the linguistic data very poorly can result. However, the ease with which features can be added is also a major advantage of unification grammars, as it is possible to maintain large amounts of information at the same time in the same structure and use whatever is useful in the context of solving your NLP problem.

Using a unification-based formalism could have some impact on the *learning problem*. The addition of feature structures may mean that there is the need to learn many more things, e.g. person and number, if these are included. While these things will need to be learned, this may be a task for a future system.

Probably one of the major advantages of a unification-based system, is that it is very *extensible*. This means that the complexity of the formalism could be increased to deal with more and more syntactic phenomena as the learning system improved. For example, features for number and person could be added. However, as has been mentioned, this extension needs to be tightly constrained.

Unification grammars also add a level of *knowledge separation*, i.e. the separation between the rules and the arguments. This is useful in the context of the separation of background knowledge (the rules) and the knowledge to be learned (the arguments). This can also be thought to be plausible linguistically and psychologically *if* the rules and the parameters capture the right universals (see Principle and Parameters theory e.g. [47]). The separation can be similar to and interlinked (especially in HPSG for example) with the separation in knowledge within lexicalised formalisms.

Finally, as shown above, with respect to *elegance*, unification formalisms have potential to be extremely compact and elegant, i.e. a small number of rules replacing a large number. However, there is also the risk of building very inelegant grammars in the detail of the arguments or features themselves. Elegance thus remains almost entirely in the hands of the grammar writer.

Overall, the conclusion is that unification grammars are an excellent formalism for capturing a great deal of linguistic information. However, currently the majority of the information contained within them is probably beyond the scope of the type of learning systems that will be developed, i.e. the concepts being learned currently are not as detailed as even simple unification grammars. Hence, unification grammars for the systems presented here remain a future goal.

#### 2.2.4 Selecting the Grammar Formalism

In summary, for the syntax learning system that is to be built, the desired grammatical formalism will be a lexicalised formalism, but not a unification formalism (although perhaps this will follow later). It should also be possible to add a stochastic component to the formalism.



Many formalisms have been eliminated by the discussion above. Standard phrase-structure grammars, Principles and Parameters approaches and Tree-Adjoining Grammars (TAG) [71] are not lexicalised (although TAG does have some of the benefits). Similarly, HPSG, GPSG and DCGs are all based upon unification and so cannot be considered. Most formalisms can be extended to have a stochastic component, so this is less of an issue.

However, this still leaves a number of possible formalisms, e.g. Lexicalised Tree-Adjoining Grammar [71] and Categorical Grammar. The Categorical Grammar formalism has been selected from these remaining possibilities, as it is deemed to be particularly elegant, although this is something of a matter of personal taste. It is also true that there has been very little work on unsupervised learning of Categorical Grammars, and so it will be interesting to explore this particular avenue.

In the remainder of this chapter, the Categorical Grammar formalism is described, along with a simple stochastic extension.

## 2.3 Categorical Grammar

Categorical Grammar (CG) [144, 118] provides a functional approach to lexicalised grammar, and so, can be thought of as defining a syntactic *calculus*. Below we describe the basic CG, which proves to be suitable for the needs of the learning system.

A CG,  $G$ , can be defined as the quintuple  $G = \langle W, C, S, R, L \rangle$ . Where  $W$  is the set of terminals, i.e. the set of words in a natural language grammar.  $C$  is the set of categories that can be assigned to words. There is a set of *atomic* categories in CG, which are usually nouns (n), noun phrases (np) and sentences (s). Sometimes the atomic categories pp and prt for prepositional phrases and particles respectively are also used. Wood [144] suggests that, while these appear as atomic categories, they can also be considered as abbreviations for the full complex categories for these constituents. It is then possible to build up *complex* categories using the two slash operators “/” and “\”. If  $A$  and  $B$  are categories then (following Steedman’s notation [118])  $A/B$  is a category and  $A \backslash B$  is a category, i.e.

**Definition 1** *Given a set of atomic categories  $\mathcal{A}$ , a CG category is defined as being a member of the infinite set  $\mathcal{C} = \mathcal{A} \cup \{A/B : A \in \mathcal{C} \wedge B \in \mathcal{C}\} \cup \{A \backslash B : A \in \mathcal{C} \wedge B \in \mathcal{C}\}$*

Informally, these can be thought of as the category  $A$  searching for the category  $B$  to complete it, e.g. the verb phrase category  $s \backslash np$  is a sentence looking for a subject noun phrase to complete it. More formally, the categories  $A/B$  or  $A \backslash B$  are functors which, when supplied with the argument  $B$  return the result  $A$ . The direction of the “slash” directionalities indicates the position of the argument in the sentence, i.e. a “/” indicates that a word or phrase with the category of the argument should immediately *follow* in the sentence. The “\” is the same except that the word or phrase with the argument category should immediately *precede* the word or phrase with this category. This is most easily seen with examples, so a set of categories are provided in Table 2.1.

$S$  is the special symbol that is used to describe a complete utterance (a complete sentence usually in natural language). This is usually the atomic category  $s$ .  $R$  is the set of rules used to combine categories. With basic CG there are just two rules for combining categories: the

Syntactic Role	CG Category	Example
Sentence	s	<i>the dog ran</i>
Noun	n	<i>dog</i>
Noun Phrase	np	<i>the dog</i>
Intransitive Verb	s\ np	<i>ran</i>
Transitive Verb	(s\ np)/ np	<i>kicked</i>
Ditransitive Verb	((s\ np)/ np)/ np	<i>gave</i>
Sentential Complement Verb	(s\ np)/ s	<i>believe</i>
Determiner	np/ n	<i>the</i>
Adjective	n/ n	<i>hungry</i>
Auxiliary Verb	(s\ np)/(s\ np)	<i>does</i>
Preposition	(n\ n)/ np	<i>in</i>
	((s\ np)\ (s\ np))/ np	

Table 2.1: A set of example CG categories

forward (FA) and backward (BA) *functional application* rules. Again following Steedman's notation [118] these are:

$$\begin{aligned}
 X/Y \ Y &\Rightarrow X & (FA) \\
 Y \ X \backslash Y &\Rightarrow X & (BA)
 \end{aligned}$$

These rules describe the applications of the categories in the functor-argument terms used above.  $X$  and  $Y$  should be considered as variables over all possible categories. Given an instance of a category  $A/B$  followed immediately by a  $B$ , then the first category acts as the functor, taking  $B$  as the argument and returning  $A$  as the result using FA. The process is similar for BA, but the functor category is looking backwards for its argument. This is most clearly seen with an example like that in Figure 2.2, where the parse derivation for “John ate the apple” is presented. Here we can see the similarity with Context-Free Phrase-Structure Grammars (CFPSG). The tree in Figure 2.3 shows the analysis which could be completed using the CFPSG in Figure 2.4. Each time either the FA or the BA rule is used with a pair of categories it is equivalent to the application of one of the binary CFPSG rules.

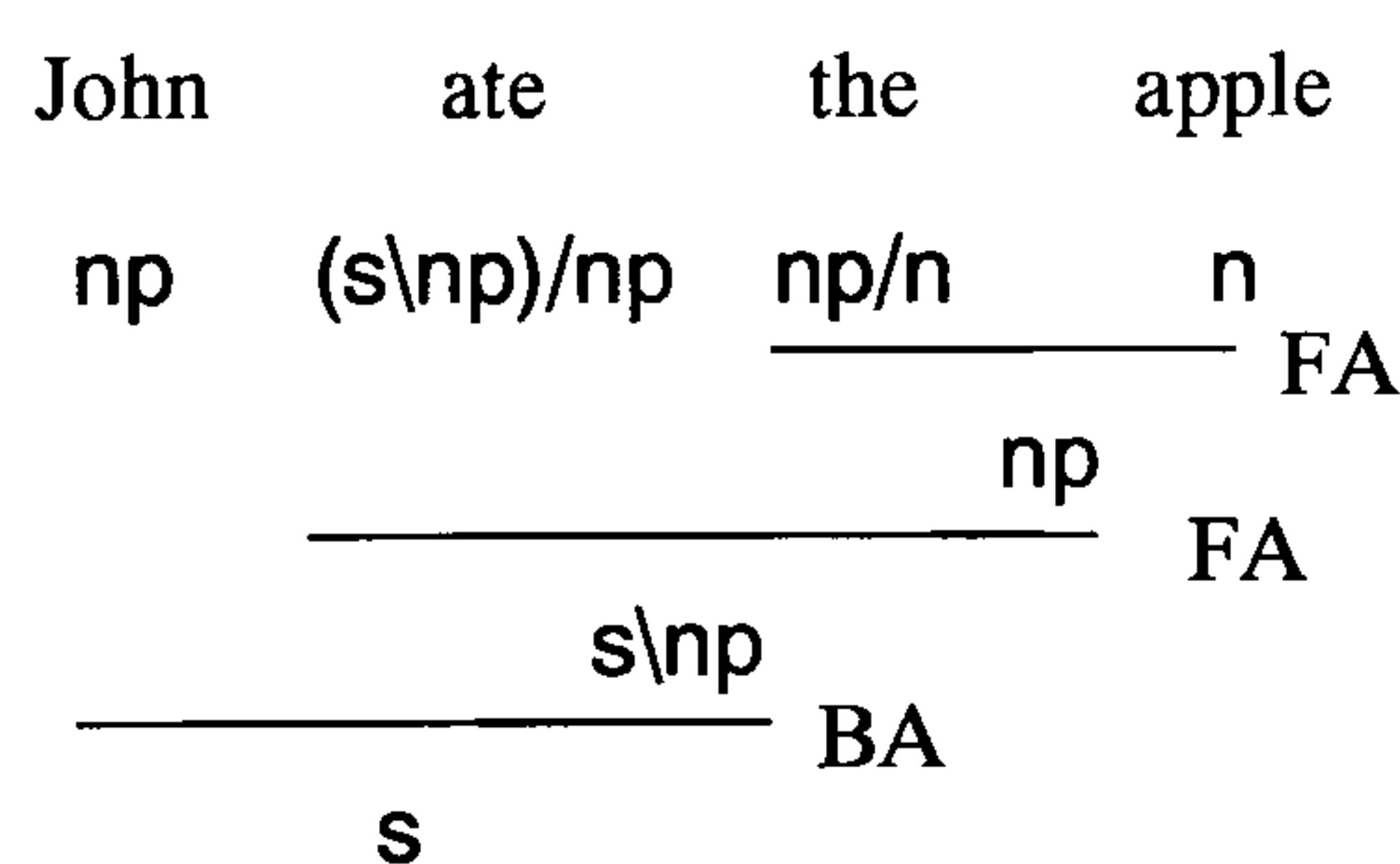


Figure 2.2: An example CG parse

Finally, to complete the definition of a Categorical Grammar,  $L$  is the lexicon. This is a set of lexical entries. A lexical entry is a word-category pair, i.e.

**Definition 2** A lexical entry is defined as a pair  $\langle w, c \rangle$  where  $w \in W$  and  $c \in C$ .

Therefore, a lexicon is a set of word-category pairs without any repetitions.

**Definition 3**  $L$  is a lexicon if it is a set of lexical entries and for two distinct members of  $L$ ,  $l_1$  and  $l_2$ ,  $l_1 \neq l_2$ .



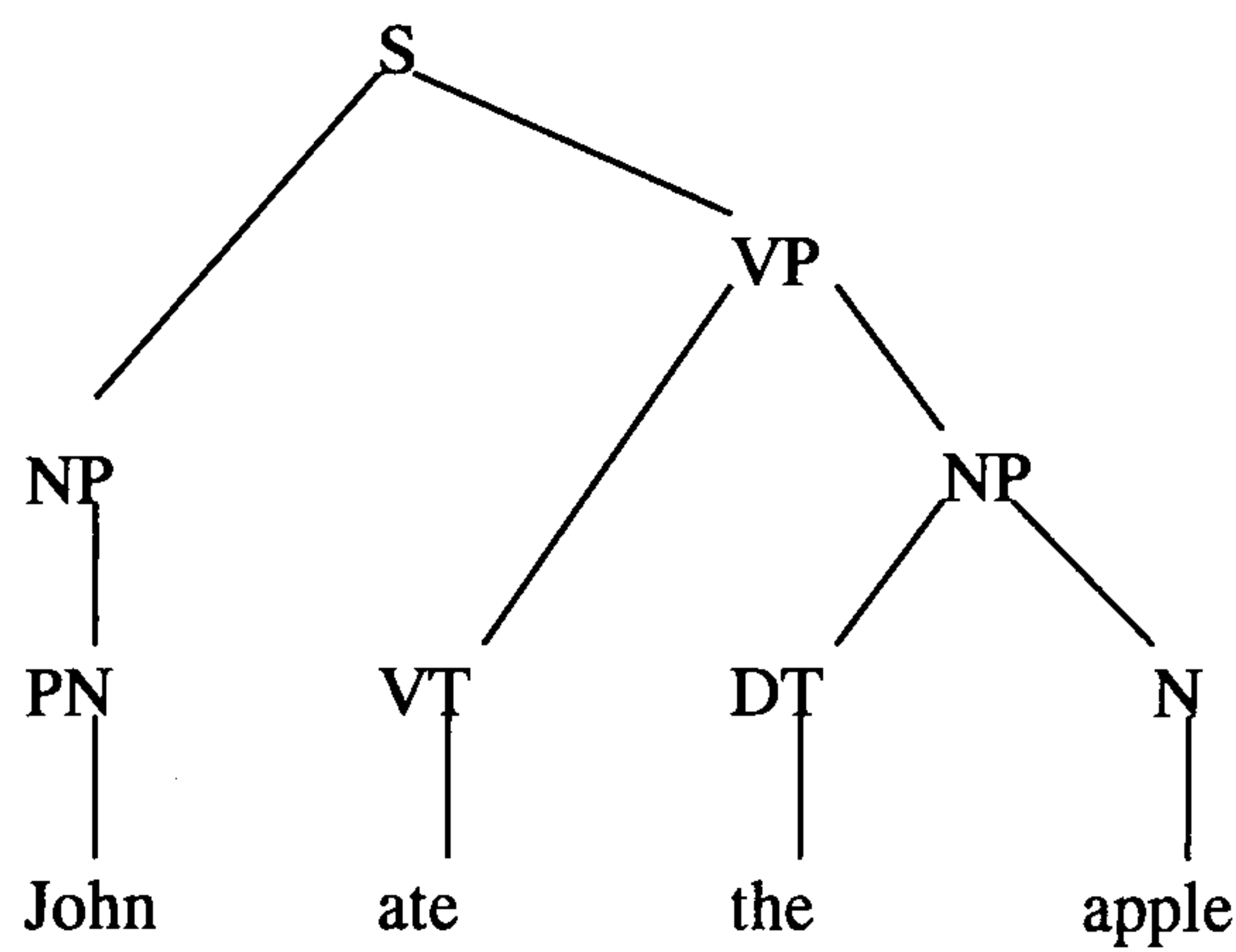


Figure 2.3: An example CFPSG parse

$S \rightarrow NP \ VP$	$PN \rightarrow \text{John}$
$NP \rightarrow PN$	$VT \rightarrow \text{ate}$
$NP \rightarrow DT \ N$	$DT \rightarrow \text{the}$
$VP \rightarrow VT \ NP$	$N \rightarrow \text{apple}$

Figure 2.4: An example CFPSG

The CG described above has been shown to be weakly equivalent to Context-Free Phrase-Structure grammars [11]. As mentioned above, while such expressive power covers a large amount of natural language structure, it has been suggested that a more flexible and expressive formalism may capture natural language more accurately [144, 118]. This has led to some distinct branches of research into usefully extending CG (see Wood [144] for an overview). An example of an extended formalism is Steedman’s Combinatory Categorical Grammar (CCG) [118], which elegantly captures certain linguistic constructs, such as coordination. An alternative method of extension, which has already been discussed, is to add a unification component. Uszkoreit [124], for example, developed Categorical Unification Grammar (CUG) to marry the categorial and unification approaches.

In this work, I have included one extra rule in the set  $R$  for building compound noun phrases (which are difficult to handle elegantly with just the functional application rules in CG [144]). It simply allows the option of adjacent noun phrases being joined.

$$np \ np \Rightarrow \ np \quad (NP)$$

## 2.4 Stochastic Models with Categorical Grammar

In this section I outline the stochastic Categorical Grammar models used within the learning system. In the first instance I formally describe the stochastic language model used (Section 2.4.1). I aim to show that it is a suitable stochastic model for both word-category pair sequences and for derivations. The models are defined rigorously, as they are important in providing some of the direction for the learning system. This model of stochastic CG is the same as that used in the learning system of Osborne and Briscoe [96].

### 2.4.1 A Language Model

The language model is best described in two stages. The first (Section 2.4.1.1) shows that the model is a well-formed stochastic model for word-category sequences. This bears some resemblance to stochastic part-of-speech tagging language models. In the second stage (Section 2.4.1.2), I show that the word-category sequence model can be used as a model for probabilistic parsing.

#### 2.4.1.1 The Word-Category Sequence Model

To define the model, we first define the concept of a *stochastic lexical entry*.

**Definition 4** *If  $W$  is the set of words in a language and  $C$  is the set of Categorical Grammar categories that may be assigned to words in that language, then a lexical entry is defined as a triple  $\langle w, c, p \rangle$ , where  $w \in W$ ,  $c \in C$  and  $0 \leq p \leq 1$ .*

Informally, a lexical entry assigns a category to a word with a given probability, e.g. the entry  $\langle \text{the}, \text{np/n}, 1.0 \rangle$  shows that the word “the” is a determiner, i.e. takes the determiner category np/n, with a probability of 1.0 – in this case the category is defined to be certain.

I can now define what the *stochastic lexicon* is in this stochastic model of Categorical Grammar.

**Definition 5** *If  $W$  is the set of words of a language, then the set of lexical entries  $L$  is a stochastic lexicon iff*

$$\forall w \in W : \sum_{\langle w, c, p \rangle \in L} p = 1$$

and

$$\forall \langle w_1, c_1, p_1 \rangle \in L, \langle w_2, c_2, p_2 \rangle \in L : \langle w_1, c_1, p_1 \rangle \neq \langle w_2, c_2, p_2 \rangle \iff w_1 \neq w_2 \vee c_1 \neq c_2$$

A lexicon is thus defined to have two constraints. Firstly, given a word that is in the lexicon, the sum of the probabilities of the entries for that word is one. That is, a probability distribution is defined for each word in the lexicon over the different categories that the word may take. For example, the word “bank” could perhaps take three categories: noun (n), intransitive verb (s\np) and transitive verb ((s\np)/np)). This should give a lexicon with three lexical entries for “bank” where the probabilities sum to one, e.g.  $\langle \text{bank}, \text{n}, 0.4 \rangle$ ,  $\langle \text{bank}, \text{s\np}, 0.3 \rangle$  and  $\langle \text{bank}, (\text{s\np})/\text{np}, 0.3 \rangle$ . If these entries did not add up to one then it would not be a lexicon.

Secondly, there should be no duplicate entries in the lexicon. By this I mean that there can be only one lexical entry for a given word-category pair. For example, if there were two lexical entries  $\langle \text{bank}, \text{s\np}, 0.3 \rangle$  and  $\langle \text{bank}, \text{s\np}, 0.1 \rangle$  in a set of lexical entries, then that set would not be a lexicon.

Given these definitions, it is possible to determine the probability of any sequence of word-category pairs by calculating the product of the probabilities of each item in the sequence. More formally, given a lexicon  $L$  for a language and the set of words  $W$  for the same language, then given a sequence of  $n$  words  $w_1 w_2 \dots w_n$ , where  $w_i \in W$  for  $1 \leq i \leq n$  it is possible



to generate at least one sequence of lexical entries, e.g.  $\langle w_1, c_1, p_1 \rangle \langle w_2, c_2, p_2 \rangle \dots \langle w_n, c_n, p_n \rangle$ , such that when  $\langle w_j, c_j, p_j \rangle \in L$  where  $1 \leq j \leq n$ , the probability for this sequence of word is defined to be  $\prod_{j=1}^n p_j$ .

Given a lexicon for a language  $L$  and the set of words in the language  $W$ , then for any sequence of words  $w_1 w_2 \dots w_n$  where  $w_i \in W$  and where  $1 \leq i \leq n$ , then there is a set of one or more possible sequences of lexical entries. The sum of the probabilities of all the sequences of lexical entries for any given sequence of words, is one.

The proof is relatively trivial. Suppose that  $p_{ij}$  is the probability of word  $w_i$  receiving category  $c_j$ , where  $1 \leq i \leq n$  and  $1 \leq j \leq m$ , given that  $m$  is the total number of categories. In this case, the sum of the probabilities of all possible sequences of word-category pairs is

$$\begin{aligned} & (p_{11} \times p_{21} \times \dots \times p_{n1}) + (p_{12} \times p_{21} \times \dots \times p_{n1}) + \dots + \\ & (p_{1m} \times p_{21} \times \dots \times p_{n1}) + \dots + (p_{11} \times p_{2m} \times \dots \times p_{nm}) + \dots + \\ & (p_{1m} \times p_{2m} \times \dots \times p_{nm}) \end{aligned}$$

Using the distributive law, each distribution can be factored out in turn, e.g.

$$\begin{aligned} & (p_{11} + p_{12} + \dots + p_{1m})(p_{21} \times \dots \times p_{n1}) + \dots + \\ & (p_{11} + p_{12} + \dots + p_{1m})(p_{2m} \times \dots \times p_{nm}) \\ & = (p_{21} \times \dots \times p_{n1}) + \dots + (p_{2m} \times \dots \times p_{nm}) \end{aligned}$$

which eventually reduces the sum to one, as required.

This shows that this oversimplified (because of the assumed independence between the word-category pairs, which is clearly an incorrect assumption) stochastic version of Categorical Grammar defines a probability distribution over sequences of word-category pairs. In other words it is a well-formed probabilistic model and can be used to assign a probability of any mapping generated between words and categories. This is clearly not the same as saying that there is a well-formed model for derivations or parses.

To clarify this model, a small worked example will be given. Suppose we have the stochastic lexicon given in Figure 2.5. Note that this is a stochastic lexicon, the lexical entries are of the form defined above, there are no entries with same word and category as any other entries and the probabilities of different entries for the same words sum to one.

$\langle \text{the}, \text{np/n}, 1.0 \rangle$   
 $\langle \text{man}, \text{n}, 0.7 \rangle$   
 $\langle \text{man}, (\text{s}\backslash\text{np})/\text{np}, 0.3 \rangle$   
 $\langle \text{saw}, (\text{s}\backslash\text{np})/\text{np}, 0.8 \rangle$   
 $\langle \text{saw}, \text{n}, 0.2 \rangle$   
 $\langle \text{girl}, \text{n}, 1.0 \rangle$

Figure 2.5: A example of a stochastic CG lexicon

Given the sentence:

“the man saw the girl”

there are four possible sequences of lexical entries (shown in Figure 2.6), which correspond to the different possible configurations of the ambiguous words “man” and “saw”. The

probabilities (the calculation of which is also shown in Figure 2.6) are can be summed in the following way:

$$0.56 + 0.24 + 0.14 + 0.06 = 1$$

which shows an example of the probabilities of lexical-entry sequences summing to one.

$$\begin{array}{rcl}
 \langle \text{the, np/n, 1.0} \rangle & \langle \text{man, n, 0.7} \rangle & \langle \text{saw, (s\ np)/np, 0.8} \rangle \quad \langle \text{the, np/n, 1.0} \rangle \quad \langle \text{girl, n, 1.0} \rangle \\
 1.0 \quad \times & 0.7 \quad \times & 0.8 \quad \times \quad 1.0 \quad \times \quad 1.0 \\
 & & = 0.56 \\
 \langle \text{the, np/n, 1.0} \rangle & \langle \text{man, (s\ np)/np, 0.3} \rangle & \langle \text{saw, (s\ np)/np, 0.8} \rangle \quad \langle \text{the, np/n, 1.0} \rangle \quad \langle \text{girl, n, 1.0} \rangle \\
 1.0 \quad \times & 0.3 \quad \times & 0.8 \quad \times \quad 1.0 \quad \times \quad 1.0 \\
 & & = 0.24 \\
 \langle \text{the, np/n, 1.0} \rangle & \langle \text{man, n, 0.7} \rangle & \langle \text{saw, n, 0.2} \rangle \quad \langle \text{the, np/n, 1.0} \rangle \quad \langle \text{girl, n, 1.0} \rangle \\
 1.0 \quad \times & 0.7 \quad \times & 0.2 \quad \times \quad 1.0 \quad \times \quad 1.0 \\
 & & = 0.14 \\
 \langle \text{the, np/n, 1.0} \rangle & \langle \text{man, (s\ np)/np, 0.3} \rangle & \langle \text{saw, n, 0.2} \rangle \quad \langle \text{the, np/n, 1.0} \rangle \quad \langle \text{girl, n, 1.0} \rangle \\
 1.0 \quad \times & 0.3 \quad \times & 0.2 \quad \times \quad 1.0 \quad \times \quad 1.0 \\
 & & = 0.06
 \end{array}$$

Figure 2.6: The possible lexical entry sequences of the example sentence

Two pertinent comments need to be made about this stochastic model at this stage. Firstly, the model is a uni-gram model, i.e. probabilities are attached to one lexical entry and no surrounding context is taken into account. This is a *very* simple statistical model. It is much more common to use models that take at least one or two of the surrounding words (or lexical entries in this case) into account. However, as has already been stated, the aim has been to use simple stochastic models as an extension to more complicated symbolic (non-stochastic) models. This stochastic extension has been used before by Osborne and Briscoe [96] for a similar task, and hence seem especially appropriate. It is also advantageous that, because the stochastic extension is simple, it is cheap and efficient to build and use the model. However, it may be useful in future to investigate more complex models, especially those that take surrounding context into account.

The second issue, which should be obvious from Figure 2.6, is that not all sequences of lexical entries will allow parses. In fact, in the example, only the first sequence will allow a parse. This issue is discussed in the next section, where the model for parsing is described.

### 2.4.1.2 The Parsing Model

Firstly, it will be necessary to discuss the rules of CG and how they will be used in the probabilistic setting. The two standard rules of functional application are repeated below along with the extra noun-phrase joining rule.

$$\begin{array}{rcl}
 X/Y \ Y & \Rightarrow & X \quad (FA) \\
 Y \ X \backslash Y & \Rightarrow & X \quad (BA) \\
 np \ np & \Rightarrow & np \quad (NP)
 \end{array}$$

These three rules need to be modified to use the word-category-pair probabilities to calculate the likelihood of particular parses. As before with the non-stochastic version, one of these rules is used to combine two categories at each step of the parsing process.



However, instead of merely combining categories, it is necessary to calculate probabilities. To accommodate these calculations the rules must be redefined with respect to category-probability pairs.

$$\begin{aligned} (X/Y, P_i) (Y, P_j) &\Rightarrow (X, P_i \times P_j) \quad (FA) \\ (Y, P_i) (X \setminus Y, P_j) &\Rightarrow (X, P_i \times P_j) \quad (BA) \\ (np, P_i) (np, P_j) &\Rightarrow (np, P_i \times P_j) \quad (NP) \end{aligned}$$

This allows us to assign probabilities during derivations. An example of a probabilistic derivation is shown in Figure 2.7. This is the same as the approach proposed by Osborne and Briscoe [96]. Osborne [95] also proposes a stochastic CG where a probability distribution is defined over the set of categories, but this seems a much less useful approach, as it does not give the distribution over sequences of words.

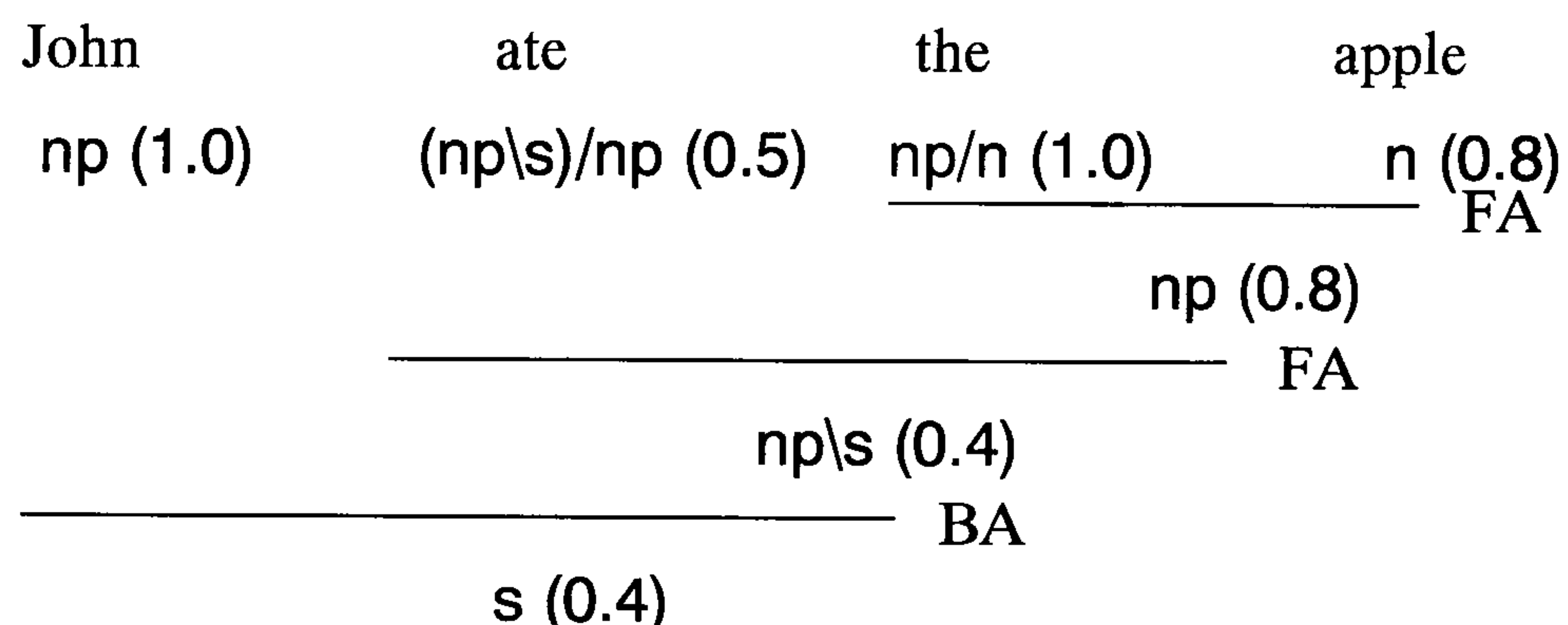


Figure 2.7: An example of a probabilistic parse in stochastic CG

As mentioned above, this probabilistic model of CG does not define a probability distribution over the derivations of examples. The distribution remains over sequences of word-category pairs. If the grammar does not allow some of these sequences of word-category pairs to be a parse, as in the example given above, then their probability mass will be lost. While this does not change the fact that the model will assign likelihoods to the parses, these likelihoods should not be considered as probabilities.

Hence, the probabilistic extension of the grammar needs to be seen as providing some stochastic constraints, which interact with the symbolic constraints that were already present in the grammar. The symbolic constraints define the possible derivations and the stochastic constraints determine the likelihood of those derivations.

Again, this approach needs to be justified. Firstly, it is again worth noting that the approach followed is simple and so matches the aim of using simple stochastic extensions to symbolic model. Secondly, this method allows cheap calculation of likelihoods, which in a learning system is advantageous. Thirdly the model has been used before in this way by Osborne and Briscoe [96] and has proved effective. Finally, as Osborne and Briscoe note [96], one could normalise the likelihoods, so that they define a distribution. If there are  $n$  parseable lexical-entry sequences for an example with likelihoods  $p_1, p_2, \dots, p_n$ , then this set of likelihoods can be turned into a probability distribution over the parses by calculating what proportion of the probability mass a sequence is, i.e. the probability of  $p_i$  where  $1 \leq i \leq n$  is:

$$\frac{p_i}{\sum_{j=1}^n p_j}$$

However, given that all the probabilities of parse sequences are not necessarily calculated by the parser (indeed in CLL only the  $n$ -best parses are calculated) then this may not be a practical solution. What this does show, is that the likelihoods assigned to a parse are directly related to a probability distribution, where the probability is conditional on the category sequence being a parse. The work of Hockenmaier [66] with respect to parsing Combinatory Categorical Grammar (CCG) is one possibility where probabilities of local trees are used to calculate the total probability. However, the calculations and the collecting of appropriate statistics would be more costly with this model, and so it seems, currently, to be more sensible to use the simpler model that has been described.

This stochastic model is perhaps somewhat simple, both in assigning probability to ungrammatical sequences (although this can be ignored if the probabilities are considered to be normalised) and in not taking into account any context apart from the current word. In future a more complex model, which considers some context, may improve the grammar. For the current purposes, however, this very simple model is acceptable. It is simple and it is therefore efficient for the calculation of probabilities in the learning system, as will be shown in Chapter 5.

### 2.4.1.3 A Comparison with Probabilistic Context-Free Grammars

The use of Probabilistic Context-Free Grammars (P-CFGs) in the context of learning systems is considered in some detail in Chapter 4. However, as P-CFGs are probably the most common way of defining syntactic structures probabilistically, then it seems appropriate to discuss their relationship to the stochastic model of CG that has just been presented.

A P-CFG, according to Charniak [33], is a quadruple  $\langle W, N, N^1, R \rangle$ , where  $W$  is the set of terminal symbols  $\{w^1, \dots, w^\omega\}$ ,  $N$  is the set of nonterminal symbols  $\{N^1, \dots, N^\nu\}$ ,  $N^1$  is the start symbol and  $R$  is the set of rules of the form  $N^i \rightarrow \zeta^j$  where  $\zeta^j$  is a sequence of symbols in either  $W$  or  $N$ . Each rule has a probability assigned to it and the probabilities for all rules with  $N^i$  on the left-hand side must sum to one. The probability of a parse is then the product of the probabilities of the rules used to make up the parse.

The P-CFG model is rather different to the stochastic CG model defined above. Again, the P-CFG model does not create a probability distribution over the set of parses for an utterance, but rather over the rules in the grammar with the same nonterminal on the left-hand side. This is perhaps most easily seen with an example.

Given the P-CFG in Figure 2.8, then there is only one possible parse for the sentence:

“John likes Mary”

which is shown in Figure 2.9. But this parse has the probability:

$$0.33 \times 0.33 \times 0.5 \times 1 \times 1 = 0.125$$

In contrast, the closest stochastic CG is given in Figure 2.10. It should be noted that, as the probabilities are based upon words, all the lexical entries have a probability of 1. Hence, the probability of the parse of the sentence (shown in Figure 2.11) is also 1, because the probability that is calculated by the application of rules is simply the product of the



S	→	NP VP	1
VP	→	Vi	0.5
VP	→	Vt NP	0.5
NP	→	John	0.33
NP	→	Mary	0.33
NP	→	walks	0.34
Vi	→	ran	0.5
Vi	→	walks	0.5
Vt	→	likes	1

Figure 2.8: An example P-CFG

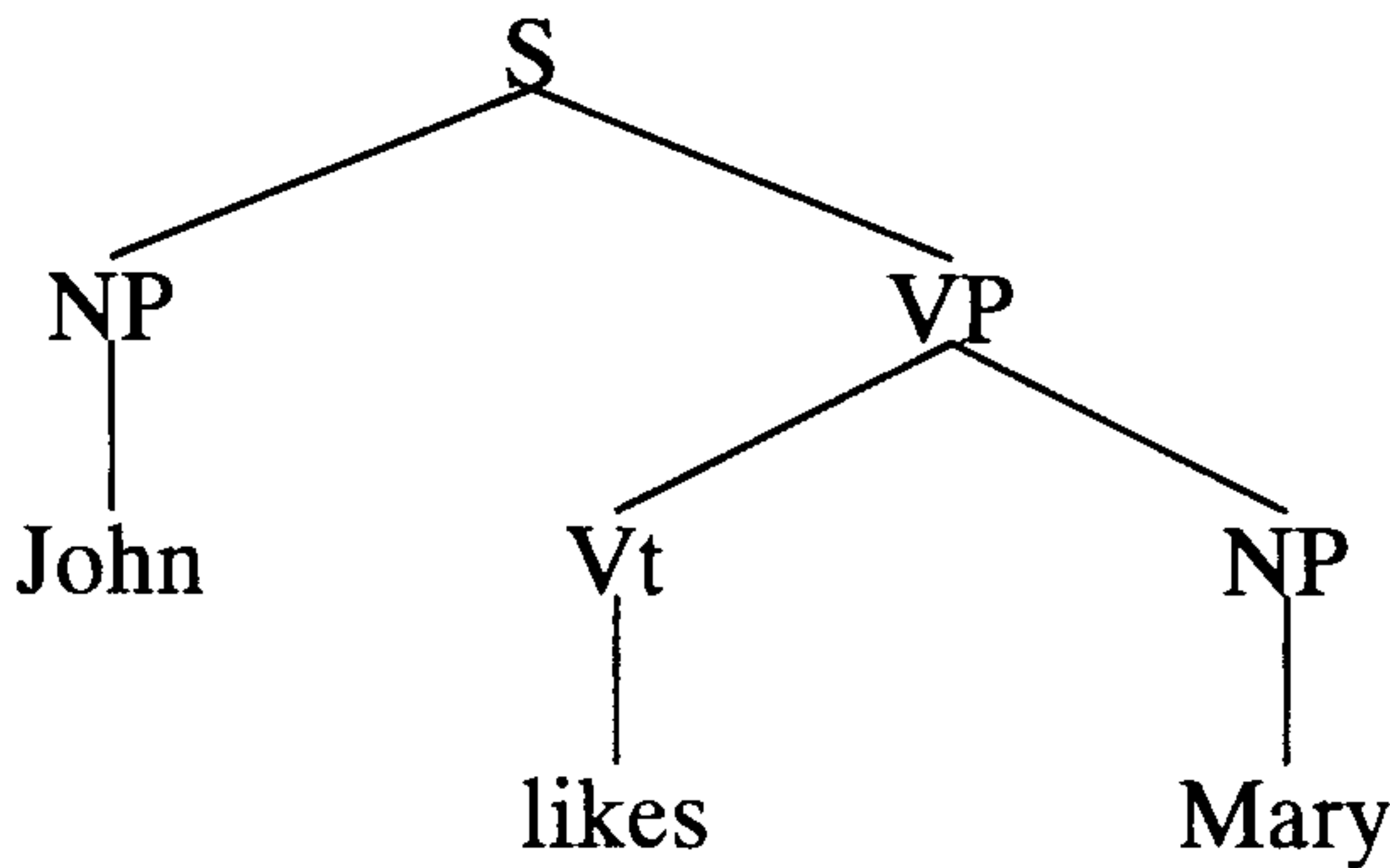


Figure 2.9: The CFG parse of the example

probabilities of the words. In this example, the stochastic CG would appear to model the situation better.

John:np	1
Mary:np	1
walks:np	0.5
ran:s\np	1
walks:s\np	0.5
likes:(s\np)/np	1

Figure 2.10: A similar stochastic CG lexicon

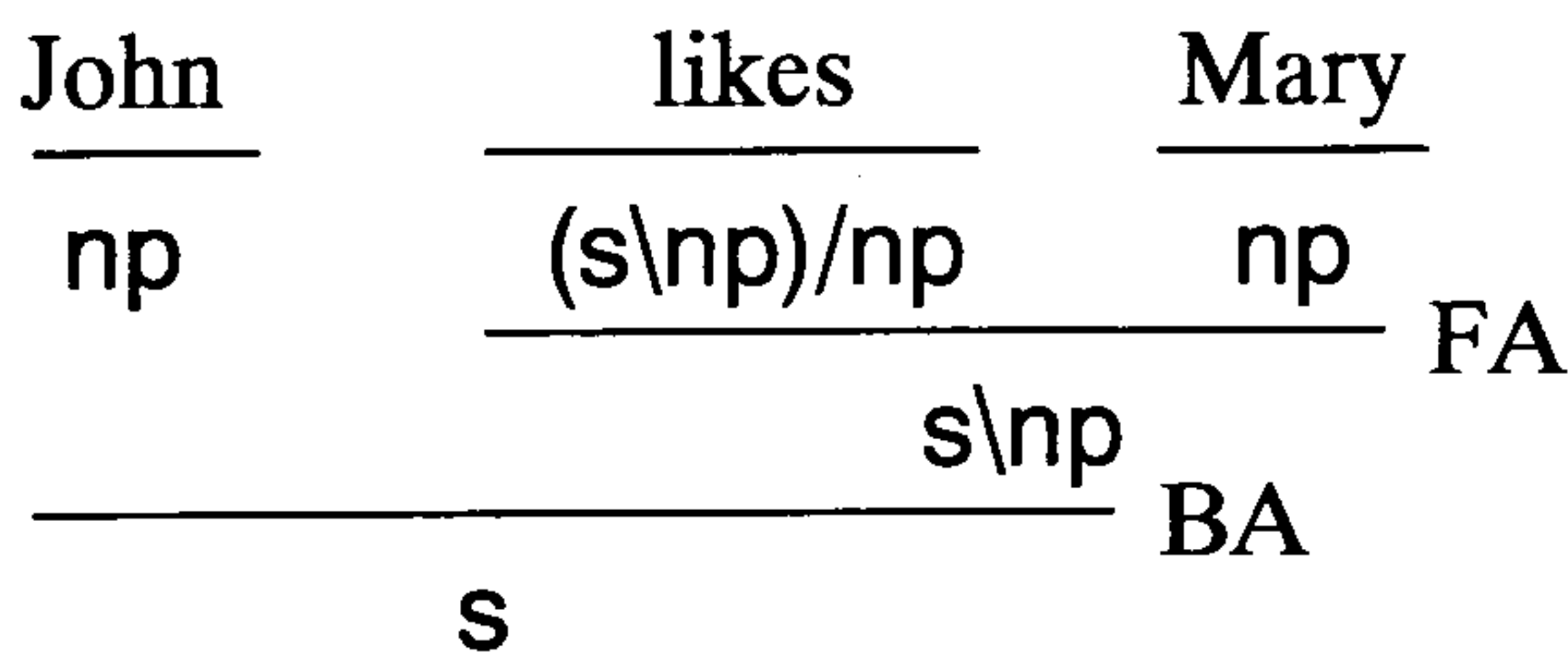


Figure 2.11: The CG parse of the example

On the other hand, there are cases where the stochastic CG also loses probability mass. For example, given the simple sentence:

“John walks”

there is only one possible parse (shown in Figure 2.12). which has a probability of:

$$1 \times 0.5 = 0.5$$

However, it should be noted that the sentence, parsed in essentially the same way with the P-CFG (shown in Figure 2.13), has a lower probability still:

$$0.33 \times 0.5 = 0.165$$

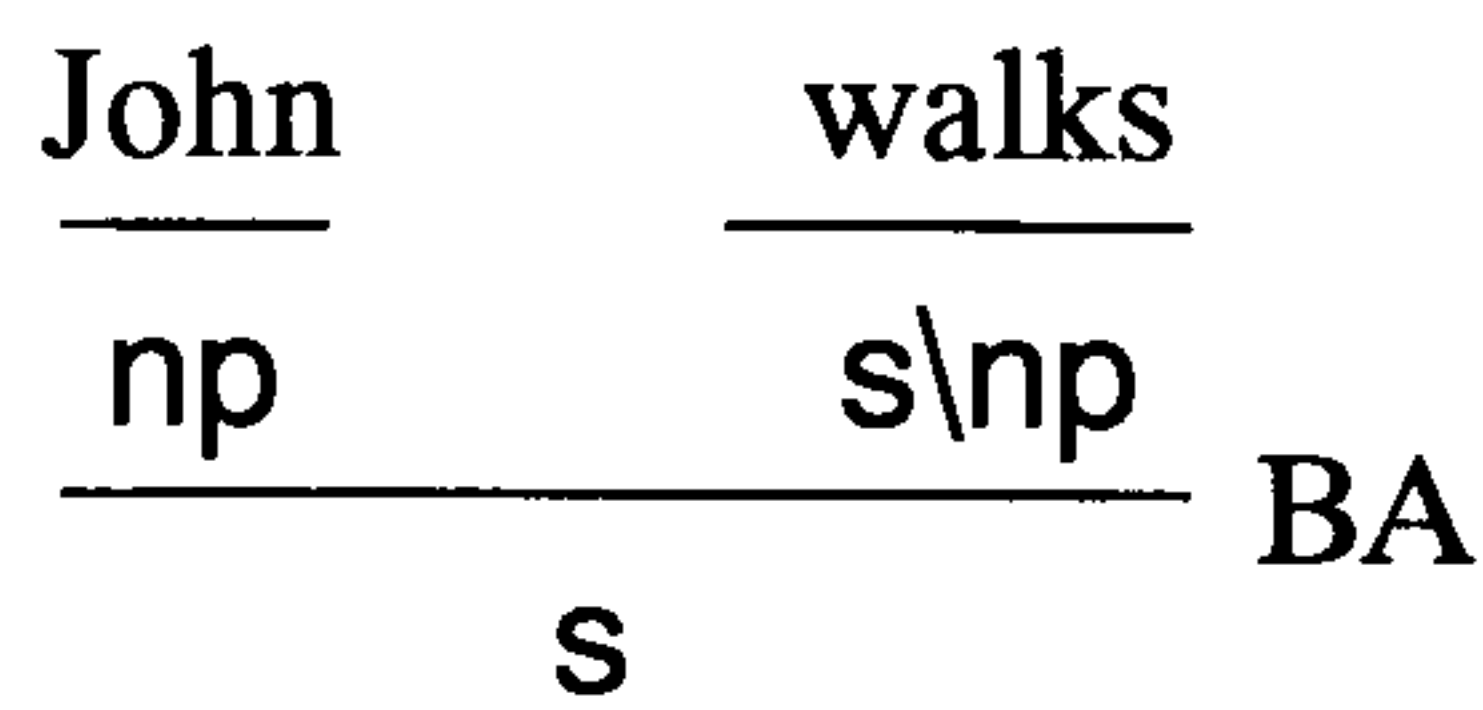


Figure 2.12: The CG parse of the example

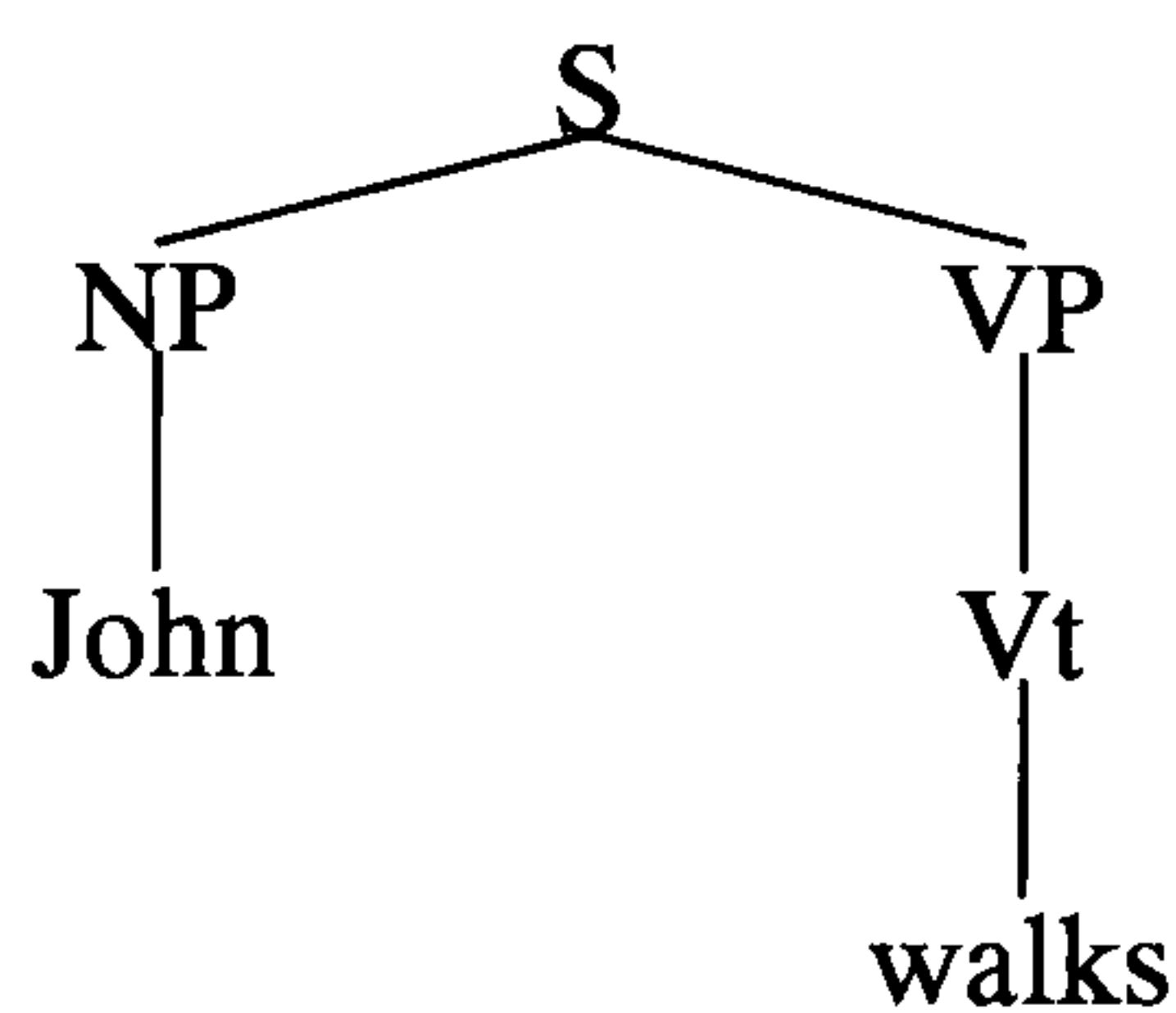


Figure 2.13: The CFG parse of the example

Hence, it can be seen that P-CFGs and stochastic CGs are somewhat different in the way that probabilities are added to the basic context-free grammar structures. The examples above suggest that attaching probabilities to lexical items and ensuring that the probability distributions are defined over words is perhaps a better way of maintaining the probability mass. There is some (rather weak) support for the stochastic CG model being better than a model which defines probability distributions over each category (as in P-CFG) rather than the word from the work of Osborne [95] and Osborne and Briscoe [96]. Results are presented on the grammar learning problem (see Chapter 4 for a discussion of their work), where a significant difference between the experiments is the type of stochastic CG used. In Osborne [95] probability distributions are defined over categories, whereas in Osborne and Briscoe [96] the probability distributions are defined over words (in the same way as they have been here). The results are somewhat better in the latter paper. It may be, that for a lexically driven model it is better therefore to have a model that assigns probability distributions to words rather than nonterminals or categories.

However, there is a weakness of the stochastic CG that P-CFGs do, to some extent, overcome. The stochastic CG is essentially a uni-gram model, as mentioned above. Hence, when calculating the probabilities of sequences of word-category pairs, no context is taken into account. P-CFGs do, indirectly, take context into account. This is achieved by assigning probabilities to rules higher than lexical rules, e.g. in Figure 2.8 the rules  $VP \rightarrow Vt\ NP$ , which has a probability of 0.5, can be considered to be assigning a probability to the situation where a  $Vt$  and an  $NP$  occur together. However, P-CFGs have generally been considered to attach such probabilities in the wrong place [44, 18] and so it is perhaps unwise to consider using it as a language model.

It is, of course, possible to think of stochastic CGs and P-CFGs in a somewhat similar



way. The stochastic CG can be thought of as a P-CFG where all the rules have a probability of 1 (although this is not a well formed P-CFG, as if there is more than one rule with the same non-terminal on the left-hand side then the probabilities will not sum to one).

In this context, it seems that, while the stochastic CG defined here is not unreasonable and is unlikely to be any less useful than a P-CFG, it may be the case that the stochastic CG will need to be modified in the future to allow more context to be taken into account.

## 2.5 Conclusion

In this chapter, the requirements for an appropriate grammar for the learning system have been considered. These requirements have led to Categorical Grammar being chosen. As a formalism it is lexicalised in an appropriate way, and (as has been shown) can be easily extended to have a simple stochastic component. It expresses the information desired at an appropriate linguistic level, but should more detail be required in future it can be extended to a unification-based formalism.

Having defined the grammatical formalism, it is now possible to continue to develop the model and finally the system that will use this formalism.

## Chapter 3

# A Computational Framework for Syntax Learning

Having determined that the system will learn stochastic Categorical Grammar lexicons, the next stage in the process is to determine a computational framework within which to design and implement the syntax learning system. The implementation of this approach is described in detail in Chapter 5.

As has already been stated, the aim is to build a practical computational system, i.e. a system that can be applied to real and useful problems. This aim will guide many of the decisions made in determining this framework. However, other issues will be noted and taken into account, particularly those related to the types of problems humans face and the sorts of methods humans appear to use to solve those problems.

Two stages are presented to describe the computational framework. The first is the definition of a general computational syntax-learning model (Section 3.1). The second is a description and evaluation of some of the types of models and methods that can be used to instantiate this model, i.e. a discussion of some of the general implementation issues with respect to the model (Section 3.2).

### 3.1 A Computational Syntax-Learning Model

In this section, the computational syntax-learning model, which is proposed and investigated in this work is described and discussed. The model is shown in Figure 3.1. Each component is described in turn in the following sections, along with some discussion as to the type of data or system that will be involved.

#### 3.1.1 Examples

The *examples* can be characterised by four features. They are:

1. lists of words,
2. all sentences,
3. unannotated or weakly annotated and



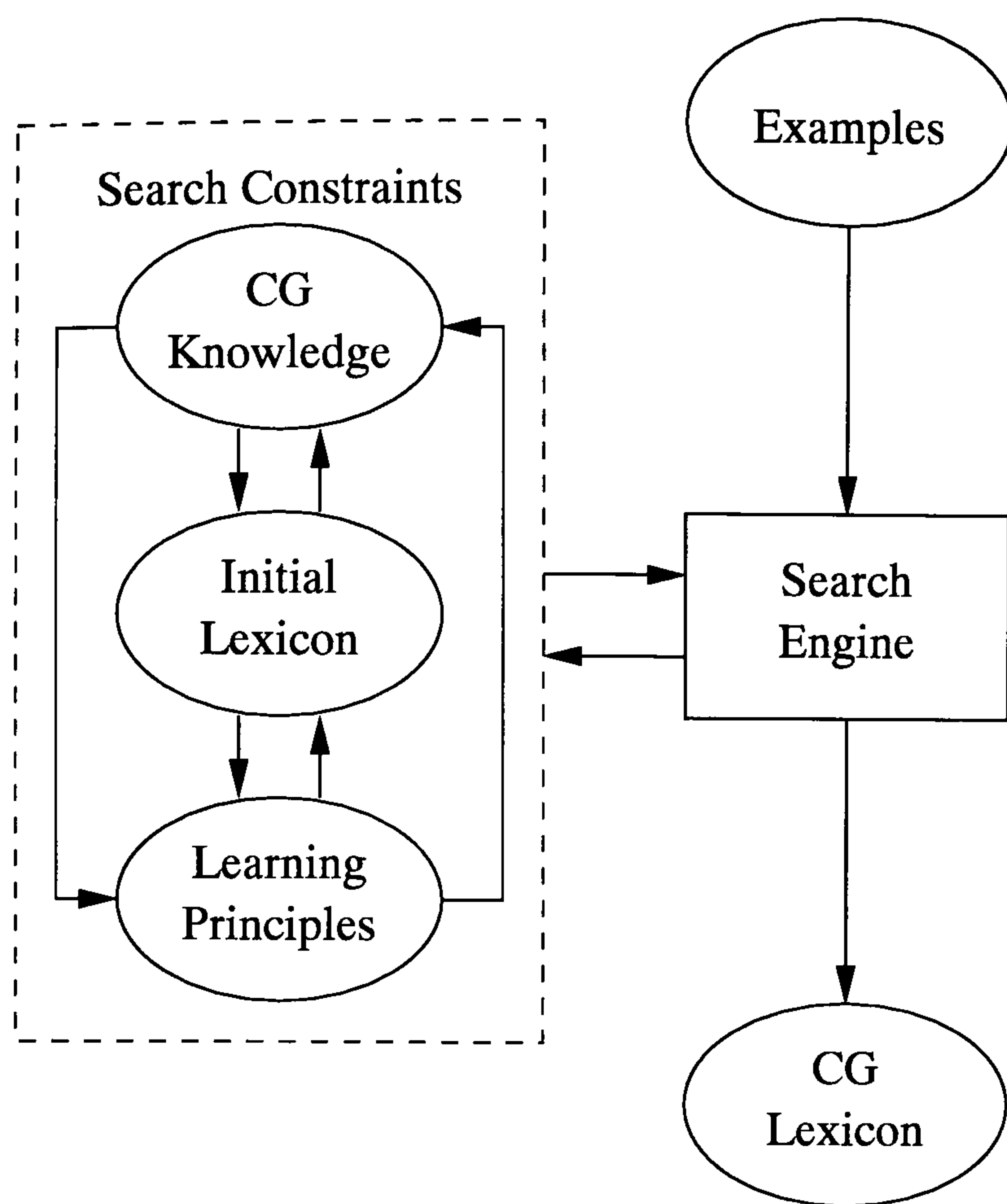


Figure 3.1: The computational model for learning syntax

## 4. all positive examples.

Each of these features is discussed in turn below.

Firstly, the examples are lists of words. This may seem an obvious fact, but it needs to be taken into account that there is no ambiguity left with respect to the word content of the examples, as there might be if continuous speech had been presented to a learning system. This is entirely practical with respect to a computational system, as there are large corpora of such text that available. However, with respect to the similarity of the problem with human language learning, it needs to be noted, that the examples presented are not in the same form as those that a child hears, unless one presumes that the child has already developed a perfect speech-recognition system.

Secondly, all the examples presented to the learner will be sentences. This provides a certain amount of useful constraining information, as the system will be able to use this fact to insist that the grammar learned should be able to assign sentence structures to these examples. Again this is reasonably practical, as the vast majority of written text is in sentences and so large numbers of examples are available. However, there are still many examples in written text that are not fully formed sentences, for example the Penn Treebank [88], which is used for some of the experiments, contains many examples that are not complete sentences, despite the fact that the text is taken from a newspaper. Again, it can be noted that in the context of a child learning language, examples are commonly not provided as sentences, but fragments of sentences. It may, therefore, be useful to extend the system in future to deal with sentence fragments and maybe even with spoken language, although it should be noted that dealing with spoken language would significantly increase the complexity of the problem, as there would be the need to deal with a large range of

phenomena that do not occur in written text.

Thirdly, the lists of words that make up the examples will either have no annotation or weak annotation. The reason for this is that naturally occurring text, of which there is usually a large amount for whatever language or domain we are interested in, does not usually come with annotation of any sort. Any annotation, e.g. parse trees or semantic form, needs to be added. The process of annotating text is time consuming and costly and it may be necessary to annotate text with all sorts of types of annotation for different tasks. Hence, we are interested in building learning systems, which do not need any annotation, or at worst need a very small amount of annotation, which is easy and cheap to do. In the work presented here we investigate using completely unannotated data and data where the nouns and verbs are marked.

The use of unannotated and weakly annotated examples has an impact on the types of learning algorithms used and the amount of knowledge that the algorithms need to be supplied with, but it should lead to more general systems that can easily be used in more situations. Hence, the aim of building a system that is practical and solves useful problems is met.

There is the further issue of what type of examples humans learn from. There is some debate with respect to this, which is described in Chapter 7, but it is certainly a strong possibility that unannotated or weakly annotated examples most closely correspond to the type of data children receive.

Finally, the only examples provided to the learner will be positive examples, i.e. examples that are considered correct. Again this is the most practical approach, as there are large numbers of positive examples of language available. It may be suggested that there are also a large number of incorrect, or negative, examples available because of the mistakes that people make, however, this is only partially correct. Firstly, negative examples need to be annotated as incorrect to be useful in a learning process, otherwise they will just be treated as positive examples. As annotated negative examples are not generally available, it will be potentially time-consuming and expensive (although the annotation is weak) to annotate a corpus of examples. Secondly, there is a difficult question about what kind of negative examples will be useful, i.e. how do we supply the right kind of negative examples for learning, given the potentially infinite number of ways of getting things wrong. Thirdly, from the psychological perspective, such examples are not available to children, and yet they are able to effectively learn language (see Chapter 7). Hence, we will pursue using positive examples only. Again, this decision has an impact on the learning algorithm and background knowledge that a learner will require, which is discussed further below.

### 3.1.2 The Search Engine

The search engine is at the heart of the learning model. The search engine performs two tasks. Firstly, there is the process of determining the best analyses for a particular example. Secondly, one of these analyses must be chosen to produce the best lexicon.

The search engine will be provided with one example at a time (thus making the learner incremental, as a human learner would seem to be). It will then search the space of appropriate syntactic analyses guided by the search constraints (see Section 3.1.4); the current state



of the system's syntactic knowledge and the information provided by analyses of previous examples (note these last two could be considered part of the current version of the CG knowledge). Hence, what is needed is an  $n$ -best parser, i.e. a parser that will return the  $n$ -best syntactic analyses on the basis of the current CG knowledge. The notion of "best" in this case will be determined by the stochastic model applied to the CG. This was the very reason for which the stochastic model was developed, as it allows us to rank parses. Hence, one place in the model where simple syntactic methods have been used to extend symbolic methods is in this probabilistic parser.

When the "best", or set of the "best", analyses have been determined, the syntactic knowledge they provide is incorporated into the current grammar to improve it and the process is repeated on the next example. In effect, the set of constraints is modified at each step by the search engine, hence, the arrows in both direction in Figure 3.1.

Again, the process of incorporating the syntactic knowledge is guided both by symbolic and statistical constraints. The aim is to build the most compressive lexicon, i.e. by some statistical measure, the smallest lexicon. This is achieved by choosing the syntactic analysis (from the  $n$ -best), which leads to the smallest lexicon. Compression is discussed in more detail in Section 3.2. This is not simply a matter of selecting the parse which uses the lexical entries, which are most similar to the current lexicon. By adding entries to the lexicon, the nature of the lexicon may be changed enough to mean that earlier examples, if reparsed, would be parsed in a different way and so have a different impact on the lexicon. Hence, the approach taken to determining the next lexicon involves trying potential lexicons and reparsing previous examples to determine any changes.

The process in CLL for performing the selection of the analysis of an example for adding to the lexicon is discussed in more detail in the next chapter. At this stage we will simply discuss the idea of using reparsing. The choice to do this occurred because of empirical evidence from early experiments when developing the system. Initially reparsing was not used, but this led to a system that did not modify early mistakes.

For example, the early system, which did not reparse, analysed all ditransitive verbs as transitive verbs, because of choices made early on. If a word was mistakenly analysed as a transitive verb early in the learning process, it was more likely to be analysed as a transitive verb the next time it was seen and any analysis of it as a ditransitive verb would only add another lexical entry to the lexicon (so not leading to a smaller lexicon). Hence, when a mistake was made it was simply repeated. The incorrect analysis of other words was not enough to push the analysis towards the ditransitive. This is, in a sense, an example of the subset problem, where a wrong generalisation has been made and there is no evidence to prevent this occurring again.

Reparsing of previous examples with compression allows this mistake to be corrected, because, so long as the ditransitive analysis is proposed as one of the  $n$ -best analyses, then compression over a number of examples containing the same ditransitive verb will eventually show that analysis as a ditransitive leads to an overall smaller lexicon.

However, reparsing is poor on two counts. Firstly, it is clearly going to be expensive. Secondly, from a psychological perspective, it is inappropriate to store and re-use examples [14]. It may also be the case that reparsing is not necessary, as will be seen from the results.



It may be the case that enough data would allow a parser to correct any mistakes. Such an approach may be investigated in the future.

### 3.1.3 The Categorical Grammar Lexicon

It will be clear from Chapter 2 that the system will use a stochastic extension of Categorical Grammar, which has already been suggested as an appropriate formalism both linguistically and for the purposes of computational syntax learning.

Ideally, the system would be provided with a set of rules for CG, e.g. forward application and backward application, and the set of atomic categories, i.e. *s*, *n*, *np*. If it is hypothesised that these may be provided as background knowledge, then the remainder of the grammar would then have to be learned. This would mean that the full set of complex categories and the mapping between words and categories would have to be learned. Furthermore, because of the use of probabilistic techniques for defining “best” analyses, it would also be necessary to learn the probabilities of the word-category mappings, i.e. the probabilistic lexicon discussed in Chapter 2. Such a lexicon would be a particularly useful product for other systems that could use the grammar to rank parses.

However, as an initial learning problem, this would be particularly difficult, requiring a system that could learn a complete set of complex categories, a complete set of mappings between words and their CG categories and the probabilities of those mappings. Hence, I am currently investigating a simpler problem. I will suppose that a complete set of language-specific categories is also available to the learner, i.e. the learner has all the categories (atomic and complex) that are needed for the words in the examples. The problem is now reduced to learning the mappings between words and categories and the probabilities of those mappings.

The choice to reduce the difficulty of the problem in this manner can be justified in two ways. Firstly, by choosing to use unannotated examples, it will be necessary to include some sort of further background knowledge to direct the learning process, or to use better learning methods. The use of background knowledge as a starting point is acceptable. Secondly, the use of a complete set of categories is a practical solution, as for any new language or domain, the only work will be building a relatively small set of categories. Compared with building a large annotated corpus, it is suggested that this is a much simpler and cheaper option.

Consequently the grammar learned by the system will in fact be a lexicon, i.e. a probabilistic mapping between words and the CG categories. In effect, words are assigned to their groupings and assigned their syntactic functor-argument structure: since CG is being used this corresponds to the semantic functor-argument structure. This is discussed in Chapter 4, but essentially the learning problem being defined lies somewhere in between part-of-speech tagging and full grammar learning.

### 3.1.4 The Search Constraints

It is clear from the above that there will need to be strict constraints on the search engine, or else searching will be huge task. The constraints can be separated into 3 types:

- the CG knowledge,
- the initial lexicon, and



- the learning principles.

However, as shown in Figure 3.1, these sets of constraints all interact and the boundaries between them are by no means well defined. The separation is useful for conceptualising the constraints and they are discussed on this basis in Sections 3.1.4.1–3.1.4.3.

#### 3.1.4.1 The CG Knowledge

The building blocks of a CG grammar are available. The combination rules (FA, BA and NP) are available (although I experiment not using NP). A complete set of language-specific (atomic and complex) CG categories is also available. The motivation for this approach is discussed above, but to summarise, this amount of knowledge is fairly trivial to construct compared with either building a lexicon, or annotating a corpus and so leads to a system that can be more easily used for other languages or domains. The rules and categories are used to define the search space, i.e. the set of possible word-category mappings that can be used with an example are those that use the predefined set of categories in such a way as to allow the combination rules to provide a legal parse for the example.

The other CG knowledge used to constrain the search space is slightly different, as it changes dynamically. It is the current state of the syntactic knowledge that has been amassed from learning on previous examples. For example, what categories have been used before, in what situations. In other words, the probabilistic lexicon that has been learned by the system up to the current point can be used to direct the future learning process. This knowledge may be somewhat incorrect, but is the basis from which the learner starts. These constraints are dynamic, as the learner builds and modifies them throughout the learning process.

#### 3.1.4.2 The Initial Lexicon

The second type of constraint which may be used (but is not in some experiments, in particular those that use the weakly annotated data) is the initial bootstrapping lexicon. A small initial lexicon is used to reduce the possible word-category pairings for examples. This lexicon provides correct information for a small proportion of the words. If an example contains some of these words, then the initial lexicon can be used to assign possible categories to these words. This in turn restricts the possible categories of the rest of the words in the example, as they must be set so that a parse can be derived. For example, given the sentence:

“the man ran”

if the word “the” had a lexical entry in the initial lexicon:  $\langle the, np/n, 1.0 \rangle$ , then the category of “man” would be restricted to  $n$  or  $X \backslash np/n$  where  $X$  could be any category, or “man” would have to combine with “ran” to produce one of these categories. The initial lexicon along with the limited set of categories that words can take can provide a strong bias for the correct analysis (particularly in such short examples). The lexicons used are closed-class word lexicons, i.e. words of categories which only have a finite number of members. In particular the closed-class word initial lexicons contain determiners, prepositions and conjunctions.

It has been suggested<sup>1</sup> that the use of such lexicons will make the learning problem trivial. While the experiments show this not to be case, it maybe useful to consider why. Consider the following example from the Penn Treebank.

“I am waiting.”

There are a large number of these short sentences, which have no closed-class words in at all and so the initial lexicon has no impact on learning with these examples. Now consider a longer example, such as the one given below.

“you’ll see the annual unravelling of it.”

The sentence has two closed-class words that would be in the lexicon: “the” and “of”. If we assume “the” is assigned the category  $np/n$  and “of” is assigned the category  $(n\backslash n)/np$  then Figure 3.2 shows a set of possible parses using categories from the fixed set of categories. The figure shows that, while this information provides some constraints with respect to the parses that are allowed, there are still a very large number of possible parses (there are more still than those shown). If we allow “of” to take the other preposition categories in the closed-class lexicon then there are even more possible parses. Similarly, if we allow other words to take closed-class categories (as is the case in some of the experiments with CLL) then there are even more possible parses. Again, if the sentence is longer, or there are fewer closed-class words, more parses are possible.

It is also useful to state that, in practice, the number of closed-class words in sentences is seldom large enough to fully determine the categories that need to be assigned to words in the sentence, i.e. the examples given above can, in some sense, be considered to be representative. Investigation of a corpus of examples of length less than or equal to fifteen words extracted from the Penn Treebank [88, 85] (the corpus PC2 – see Chapter 8), give the results in Table 3.1. In this “No CCW” is the number of sentences with no closed-class words and “Average CCW” is the average number of closed-class words per example. The table shows the results for the examples which have length (in words) two, three, four and five and for the entire corpus (where the average example length is 9.56 words). Closed-class words were identified using the largest initial lexicon used in the experiment (see CCW2 in Appendix B). Hence, the results in Table 3.1 are the worst in terms of the initial lexicon trivialising the problem and would appear to show that it is very unlikely that the initial lexicon will trivialise the problem, as examples contain a relatively small number of closed-class words.

### 3.1.4.3 The Learning Principles

In this section the type of mechanisms that are used to constrain the search are discussed. Two statistical learning principles work alongside the symbolic search. These are:

1. maximum likelihood, and
2. compression.

---

<sup>1</sup>Personal communication with Mark Steedman, 2002





Examples	No. of Examples	No CCW	Average CCW
2 word	13	13	0
3 word	121	95	0.21
4 word	190	75	0.66
5 word	374	75	1.01
All	5000	341	2.83

Table 3.1: Closed-class word statistics for PC2

While it would be possible to use more rigorous learning paradigms, the aim has been to use simple statistical methods along with some general symbolic principles (the parsing). The learning paradigms are discussed in the next section along with why these learning principles are selected. Here we restrict the discussion to the type of statistical learning principles that could have been used and those that have been used.

Stochastic language models have been used commonly and effectively for computational linguistics for some years now, e.g. [33, 84, 44], particularly in the area of parsing. Following on from this research, the aim is to use the stochastic CG formalism defined in Chapter 2 to guide the search for a good CG lexicon. The grammatical information already learnt, i.e. the probabilistic lexicon, along with the initial lexicon, is used to determine likely analyses for new examples. While early in the learning process this information is likely to be lacking and inaccurate, the further the process continues, the more accurate it should become.

In general, the aim of any probabilistic learning principle is to choose the hypothesis,  $H$ , which has the highest probability given the data  $D$ , i.e. the aim is to maximise  $P(H|D)$ . More specifically, the parser should select the highest probability parse (the hypothesis) given the sentence (the data). According to Bayes' rule:

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)}$$

The problem with this rule is that we need some prior distribution for the probabilities of hypotheses, which is not usually available. However, many computational learning methods can be seen as an approximation to calculating  $P(H|D)$  with Bayes' rule.

One approach to approximating this calculation is to use the *Maximum Likelihood* (ML) principle [81], where the probability of the sentence given a particular analysis is maximised, i.e. where we have data (the sentence)  $D$ , the hypothesis (the parse) selected,  $H$ , should maximise  $P(D|H)$ . Hence, we are maximising a part of the right-hand side of Bayes' rule and using this as an approximation to calculating  $P(H|D)$ . This type of approach is used very commonly with stochastic language models (see for example Charniak [33] and also Chapter 4) and is commonly used in high quality probabilistic parsing [44, 84]. Hence, it seems a sensible principle for controlling the ranking of parses. An especially important advantage of this approach is that it uses a very simple calculation to determine the best hypothesis, hence the aim of using simple stochastic methods is maintained.

Possible alternatives to the Maximum Likelihood principle within a stochastic context would be the *Maximum Entropy* (ME) principle [81] or the *Minimum Description Length* (MDL) principle (also known as the *Minimum Message Length* (MML) principle [82]), both



of which are widely used within machine learning circles.

The ME principle is similar to the ML principle, but an estimation of the prior distribution of the hypotheses (i.e. the prior distribution of the possible parses) is used. If  $p_i$  is the estimated probability of a hypothesis  $H_i$ , then the distribution over all hypotheses  $p_1, \dots, p_k$  is estimated to be the distribution, which maximises the entropy function.

$$H(p_1, \dots, p_k) = - \sum_{i=1}^k p_i \ln p_i$$

With this estimated distribution it is possible to calculate  $P(D|H)P(H)$  and thus calculate more of the right-hand side of Bayes' rule and so approximate  $P(H|D)$ . In the case where nothing is known about the prior probability distribution, then the uniform distribution maximises this equation and so the approach reduces to that of ML. However, where extra constraints are available, such that the type of distribution is restricted, this approach becomes useful. According to Li and Vitányi [81], the ME principle has been effective in some circumstances that the ML principle has not. However, as calculating the probabilities using the ME principle is more complicated and as the ML principle has been used effectively for parsing already, it seems more sensible to use the ML principle.

Finally, with respect to possible learning principles for the parser, we consider the MDL principle. This states:

“The best theory to explain a set of data is the one which minimises the sum of: the length, in bits, of the description of the theory; and the length, in bits, of data when encoded with the help of this theory.” [81]

This can again be shown to an approximation to Bayes' rule (see Li and Vitányi [81]). The intuition behind the principle is that there is a need to balance two issues when generalising. Firstly, a hypothesis that describes the data it has seen too precisely will take noise in the data as part of the theory and new data is likely to be covered inaccurately by the hypothesis. Hence, a principle that compresses solely the data may over-fit to the data it sees. On the other hand, a principle that only compresses the hypothesis is likely to become over-general, even trivial, so that it will cover all the data and much more that is incorrect besides. Hence, a balance between these two issues is sought by compressing a combination of the data and the hypothesis.

In the context of a parser choosing the most likely parse, the MDL principle would require the minimisation of the size of the parse, or perhaps the lexical entries used by the parse (the hypothesis) plus the size of the parse encoding of the data. This approach is somewhat over-complicated for a system of ranking parses.

Hence, having considered these possibilities, it seems most sensible to use the ML learning principle with respect to the parser, as it is both simple and well-tested in the area. So, the parser will return the  $n$  hypotheses (parses) that maximise the probability of the data given the hypothesis. In practice, this means selecting the parses that give the highest probability according to the parsing model described in Chapter 2.

In early experiments, simply selecting the most likely parse in this approach did not allow the learner enough freedom to consider the impact of the parses upon the final hypothesis

we are interested in, which is not the parse of a particular sentence, but the probabilistic lexicon that is being built. Hence, given the set of parses produced with the ML parser, the next step is to select the one that modifies the lexicon in the best way. So, there is a two stage learning process and two learning principles are used.

The second principle that is used is that of compression. The idea of compression-as-learning has a long history [81] and has been used in a variety of circumstances. The intuition behind the approach is that a compressed form of some data must, to some extent, have extracted some general principles, while still keeping all the data intact. From a syntax point of view, a grammar can be thought of as a compressed form of a language, as, if it is an accurate grammar, all legal strings in the language can be generated from the grammar.

Compression is used within CLL instead of other methods of learning (see the discussion in Section 3.2) for the following reasons.

1. Compression is a simple and elegant way of expressing generalisation from large amounts of data.
2. Compression allows a simple stochastic extension to control symbolic methods of learning lexicons.
3. Compression is flexible, in that it is easy to define appropriately for a given problem.
4. Compression has been used successfully in the past for learning natural-language syntax (e.g. the work of Wolff [140, 143, 141] and Osborne and Briscoe [95, 96]).
5. Compression can be thought of as an appropriate model of human learning [142].

The work of Wolff [140, 143, 141, 142] has investigated the idea of compression for learning natural-language syntax and also as a model of the human brain. His approach is to balance two measures: compression capacity (CC) and the storage cost of a knowledge structure (Sg). CC is a measure of how much the data has been compressed. Sg is a measure of how small the hypothesis compressing the grammar is. Wolff states that the balance between these two measures is not known *a priori* [143], however, his approach is that, for a given knowledge structure (Sg), one should maximise the compression of the data (CC). The specific systems Wolff uses are discussed in some more detail in Chapter 4. Obviously, Wolff's approach is very similar to that of MDL, with the main difference being that the combination of data compression and theory compression is not predetermined.

Following this, MDL learning, which was discussed above with respect to selecting the most likely syntactic analyses, needs to be considered with respect to choosing appropriate lexicons. In the context of a grammar/lexicon learning system then the hypothesis can be thought of as the grammar/lexicon and the data encoded by the theory would be either the set of parse trees or the set of lexical categories assigned to sentences. This approach has been pursued by Osborne and Briscoe [96] learning CG lexicons. They encode parse derivations and the lexicon on the basis of word-category pair probabilities. The advantage of MDL, as mentioned above, is that it balances two constraints: covering the data and compressing the theory. This balance prevents both over-generalisation and over-fitting. However, this may not be such an advantage in the context of CLL. The use of ML in the parsing stage means



that the hypotheses presented to this compression stage have already have their probability maximised with respect to the local data (the particular example). Hence, we have already, used a learning principle that is well-known for over-fitting to the data. In this case, the ML principle will over-fit to the local data, i.e. the specific example, in the context of the lexicon, but given that the lexical entries for our hypothesis are derived from the candidate solution, this over-fitting can have an impact of the quality of the hypotheses. Hence, it would seem wise to select a balancing principle that had a tendency towards generalisation.

The general approach has been to use two of the simplest methods of compression to test if they are effective. The exact calculations involved in these two methods are given in Chapter 5. The first approach is simply to compress the number of word-category pairs in the lexicon, i.e. to compress the size of the hypothesis.

The second approach is less easy to define in terms of compressing the data encoded by the hypothesis or the hypothesis. The idea is to compress the lexicon (the hypothesis) with respect to the frequencies of lexical entries (related to the data). Osborne and Briscoe [96], in their characterisation of the problem, define this as compressing the size of the lexicon, but this is the probabilistic lexicon. This would appear to be, in some way, incorrect, as there is at least an element of the data taken into this measure of the lexicon length.

However, it is possible to view this in an alternative way. If the data encoded by the hypothesis is considered to be sequences of lexical entries rather than parse trees, then compressing the data with respect to the hypothesis is the same as compressing the lexicon taking the frequencies of lexical entries in the data into account, under the assumption that the ordering of the data is not being used in the compression. In this formulation, it is the data that is being compressed with respect to the theory and so should cause over-fitting to the data.

There is, in fact, a further way of characterising this approach with respect to what is being compressed. If the hypothesis is thought to be the word-category pairs of the lexicon and the data encoded by this hypothesis is thought of as the set of categories for each sentence (again ignoring ordering information), then this compression metric compresses both the data and the theory. In this sense, compressing the lexicon with frequency information taken into account can be considered as an approximation of MDL (an approximation because the theory and the data are not actually separated), as both the hypothesis and the data encoded by the hypothesis are compressed. While this discussion highlights the awkward nature of characterising the problem, this final characterisation seems to be the fairest, as it is clear both the data and the theory are involved in the compression.

Investigation of these two versions of compression will show whether it is necessary to include further compression of the data and whether a more formally defined MDL approach should be investigated in the future. Alternatively, it may be interesting to look at implementing an approach a little more like Wolff's [143], where it is not predetermined how data and theory compression are combined. However, at the current stage of investigation, these simple compression metrics correspond with the desire to use simple stochastic methods.

The final issue that needs to be discussed with respect to the learning principles, is an implication of using compression on the lexicons (whether probabilistic or not). The method of reparsing to correct earlier mistakes, which has already been discussed, is due in part to

the use of compression. The generalisation that is needed to get compressive lexicons over a variety of examples means that reparsing must occur to evaluate the impact of different category assignments to an example. Such reparsing, as mentioned before, will, for example, allow CLL to initially misclassify a ditransitive verb as a transitive verb and then to find that the most compressive lexicon over a range of examples will require that the verb be re-classified as a ditransitive verb.

Hence, there are two learning principles used within this model: Maximum Likelihood in the parsing, compression in selecting the smallest lexicons along with reparsing to ensure the removal of early error and to allow generalisation over more than one example.

#### 3.1.4.4 Cognitive

Two types of what might be considered to be cognitive constraints have been considered when designing the learner:

- processing constraints;
- syntactic analysis constraints;

The first of these is a set of constraints that limit the amount of processing that will be completed. For example, when searching for the best analysis of a sentence, questions arise as to how much of the search space should be investigated and how many possible analyses should be found for comparison. It seems wise to suggest that there should be a bound upon these areas of search. Similarly when proposing categories for a word, there should be a limit on complexity, this may be achieved by some of the constraints above in part, but it may also be useful to use the stochastic models to determine these simplifying strategies.

Secondly, it is possible to apply some of the psycholinguistic constraints that have been applied to human sentence-processing, in particular the process of determining possible syntactic analyses. Such constraints include approximate incrementality (dealing with each word as it is presented) and approximate determinism (not building any syntactic structure that is not used in the final analysis) [86, 87, 46]. These constraints may be used to limit the amount of work involved in producing syntactic analyses.

While, as is discussed in Chapter 7, the model is not entirely psycholinguistically plausible and indeed these constraints have not all been implemented as yet, it is worth considering how some of the psychological information that we have may be useful.

## 3.2 General Implementation Issues

In this section the aim is to discuss some of the computational and theoretical issues that have an impact on the implementation of the model presented. These issues will currently be treated in a general way, as the specifics of the implementation are discussed in Chapter 5.

The issues are grouped into three areas that will have a large impact upon the type of system that is built. These are symbolic learning, machine learning and stochastic mechanisms. Each of these is discussed in turn below.



### 3.2.1 Symbolic Learning

Symbolic learning systems aim to build symbolic concepts and knowledge from the data supplied to them, e.g. they induce rules. This is in contrast to sub-symbolic systems, such as statistical methods and neural networks, where the aim is usually to develop numerically-based models, maximising or minimising some specified function that measures the quality of the learned system.

There are a number of advantages of symbolic learning over sub-symbolic learning. Firstly, the type of concepts to be learned are often somewhat inherently symbolic. In the case of learning syntactic information, the examples are themselves a set of symbols (words and combinations of words). It is therefore most natural to express this knowledge in a symbolic way. Secondly, learning symbolic concepts gives very perspicuous results, i.e. it is easy to read and understand set of grammar rules, which is in contrast to a sub-symbolic system which commonly returns a model built of a set of numerical values. A further advantage of perspicuous results is that they can be evaluated more easily.

Symbolic learning methods do have a number of disadvantages however. Firstly, as with all purely symbolic systems, they tend not to be robust, both in the learning process and in the results they return. If the examples are noisy or inconsistent then the systems commonly fail or return poor answers. Secondly, they tend to be more difficult to build, as they commonly require complex knowledge representation and induction methods. Finally, they are commonly inefficient, as they have to manipulate much more complex entities than non-symbolic approaches.

Given that the aim is to build a syntax-learning system, it seems clear that the knowledge to be learned is most naturally expressed symbolically, and so some form of symbolic learning should be pursued. However, with respect to robustness, it will be necessary to use more than just symbolic learning.

### 3.2.2 Machine Learning Issues

In the context of Machine Learning (ML) research, a number of settings have been developed depending on the type of information available. From the perspective of practicality, it has been determined what information is available and so what kind of setting will be used. In Sections 3.2.2.1 and 3.2.2.2 the impact of these choices on the learning algorithms is discussed.

There has also been a lot of work on theoretical models for ML. Some of the most common are discussed in Sections 3.2.2.3–3.2.2.6. These models are sometimes a little restrictive for building large-scale learning-systems for real problems, or they define a model that is inaccurate to the problem being solved, but they do provide some useful computational insights.

#### 3.2.2.1 Supervision in Learning

Supervision in learning essentially involves the annotation of the data from which learning occurs. Kehler and Stolcke [75] describe supervised learning to be where:

“models are trained from data annotated with the target concepts to be learned”

In the context of syntax learning, this means examples of utterances annotated with a full syntactic structure and labelling.

Syntactic annotation has the obvious advantage of being able to abstract the required concepts from data containing much more information and perhaps not surprisingly can lead to more efficient methods than if that information was not present. It is also especially useful as the annotated examples provide a definite target, e.g. a grammar learned from annotated data can be checked by comparing how it would perform on a held-out test set of the annotated data.

However, there are some particularly serious disadvantages. Firstly, syntax learning and many other learning problems simply do not naturally occur as supervised problems. This perhaps leads to the second problem, which is that annotated resources are generally not available and are expensive to build [75].

An unsupervised learning system works from unannotated examples. In the case of syntax learning, these can be considered to be plain text. The advantages and disadvantages are to some extent the opposite of those for supervised learning. Unsupervised learning is much more practical and there is obviously a large quantity of data available. However, unsupervised learning algorithms are hard to build and the results they generate are hard to evaluate, even when annotated resources are available.

In between these extremes there are various degrees of weakly supervised learning. In these cases, the learner receives a certain amount of information with the examples from which to learn, but is not supplied with a set of data completely annotated using the target concept. For example, syntax-learning systems may use part-of-speech sequences rather than word sequences from which to learn. Hence, some syntactic information is included, but not information from the full concept. Alternatively, a small amount of fully annotated material may be used to bootstrap a learning process before applying it to completely unannotated material.

In the system presented here, both unsupervised and weakly supervised approaches are presented (Chapters 5 and 8). Both approaches use a certain amount of syntactic knowledge. In the unsupervised case, the examples do not contain the knowledge, but there is a small initial lexicon. In the weakly supervised case, noun and verb word groups are indicated by an annotation on the corpus. Both these approaches to some extent imitate the psychological environment described in Chapter 7, while still providing some bias for CLL.

### 3.2.2.2 Positive-Only Learning

A second issue which affects the learning setting, is that only positive examples are to be provided to CLL.

The choice of a positive-only learning approach means that the system must in some way address the well known *subset problem* [47, 80]. When only positive examples are presented to a learner then it is possible to postulate a grammar that will cover all these positive examples, but it can also cover a lot of examples that are not in the language, i.e. the target grammar is a subset of the hypothesised grammar.

If negative examples were available then they could be used to remove hypothesised grammars that over-generate simply by removing all grammars that cover any negative



example. However, in a positive-only setting this is not possible. Culicover [47] suggests that there are two ways of dealing with the subset problem in a positive-only setting. The first is to constrain the grammatical formalism and the second is to provide the learner with strategies to prevent over-generating hypotheses.

CLL is designed to combine these solutions to the subset problem. The CG formalism puts strong restrictions on the type of grammar (complex lexicons) that can be learned. This along with the other background knowledge is combined with syntactic and probabilistic constraints in the learning algorithm to remove over-generating models, (e.g. removing inappropriate lexical entries by compression).

### 3.2.2.3 Identification in the Limit

The *identification-in-the-limit* framework was defined formally by Gold [60]. The framework requires the learning algorithm to be tractable and when it is given an infinite sequence of classified examples (i.e. a sequence of examples annotated as to whether they are positive or negative) the algorithm will, in finite time, converge on the correct result.

This framework was originally defined with respect to learning natural language and in those terms, any algorithm that performs identification in the limit, will have the provable property that, given an infinite sequence of annotated examples, a grammar will be hypothesised until the correct grammar is converged upon. In other words, supplying more examples at this point, will not change the grammar. Within the framework there are two important theorems.

**Theorem 1** *No super-finite class of languages can be identified from a positive presentation*

where a super-finite class of languages is one that contains all finite languages and at least one infinite language, and a positive presentation is the set of all positive examples.

**Theorem 2** *Any enumerable class of recursive languages may be learned from a complete presentation*

where enumerable languages are context-sensitive, or less powerful, and a complete presentation is the set of all examples.

This framework maintains a number of the features that we desire when considering the natural language learning problem. For instance, we present examples in a sequence to the learner, which then gradually converges upon the target grammar.

However, as Bertolo [13] states it is both “too stringent and too liberal”. It is too stringent on two counts. Firstly, the learner is expected to learn the language when presented with an entirely unordered and unstructured environment. As Bertolo points out, this could be like expecting a child to learn English from legal and technical texts. Secondly, the framework defines an exact target language that the child must learn. It is clear from the variation between languages over time and area, that this cannot be the case. For language to change, it must be acceptable to learn languages slightly different from those of the environment.

The framework is perhaps also too liberal, as the learner need only learn the language eventually. Practically, this is problematic, as the sort of systems that we intend to build

need to learn the syntax of the language, or at least a close approximation, within more stringent time constraints. This rather limited restriction of learning the grammar eventually is not really supported by the psychological evidence either where, syntactically, it seems all humans have converged to their grammar by adulthood at least.

From this discussion it would seem this framework is, in a number of ways, inadequate for describing the setting in which natural language is learned. Next we look at a more recent framework that attempts to address some of these problems.

#### 3.2.2.4 PAC Learning

The Probably-Approximately-Correct (PAC) framework, defined by Valiant [125], is often described as the *Theory of Learnability*, as it gives conditions for a set of concepts to be theoretically learnable from a set of instances. Luger and Stubblefield [83] give a good definition of these. If  $C$  is a set of concepts and  $I$  is a set of instances (or examples) to be divided into positive and negative then:

“ $C$  is learnable if there exists an algorithm with the following properties:

1. The execution time for the algorithm is polynomial in the size of the concept learned, the number of properties examined, and the adjustable parameters,  $\epsilon$  and  $\sigma$ .
2. For all distributions of positive examples, the program will, with probability of at least  $(1 - \sigma)$ , produce a concept  $c \in C$  such that  $c$  fails to correctly classify instances in  $I$  with probability of less than  $\epsilon$ .”

Hence the meaning of PAC, as this method ensures that there is a high probability that the algorithm can learn concepts that are a good approximation to the correct concepts. Both  $\sigma$  and  $\epsilon$  are parameters taking values greater than or equal to zero and less than or equal to one. These parameters are used to determine the boundaries for how probable it is ( $\sigma$ ) that the learned concept fails with a given probability ( $\epsilon$ ).

The improvement over Identification in the Limit is clearly that the language need not be learned exactly. Moreover, as Bertolo [13] shows, it is possible to give a precise estimate of the amount of data required to learn the grammar, which is a significant advantage, practically, over Identification in the Limit.

The main problem with PAC-learning seems to come once again with the definition of correctness for a target concept. Condition 2 above makes it clear that to show a problem is PAC-learnable it must be shown that there is an algorithm that can produce a concept (in this case a grammar) that, within a specific range of probabilities, will classify examples correctly. If it is not possible to accurately describe a concept this means one cannot prove a PAC-learnability result. It seems clear that it is not possible to define the concept of the “correct” natural language grammar, as we discussed in the previous section. Due to the way in which language changes, people who speak the “same” language do not have exactly the same grammar. It is clear that a child may not even speak the same language as his or her parents. This is probably due to the slightly different environments in which the language is learned. However, in practice, it may be possible to define some notion of correctness. e.g.



assigning the same structures as some defined standard, but as this would be a somewhat inaccurate notion of correctness, this may not be very useful.

### 3.2.2.5 Query Learning

The final framework that has commonly been used is that of *Query Learning* where the algorithm may query an *oracle* with simple questions to which it is able to supply the answers. This framework was defined by Angluin [6].

The queries are generally very basic, such as, whether a particular example is within the set of positive examples or not. Practically, it would be exceedingly difficult to provide an oracle to answer questions about the syntactic correctness of examples. It would require a complete grammar, or a complete set of possible examples labelled as to whether or not they are correct, to be available to the oracle. Note that the view described in this thesis is not that subscribed to by all researchers. Adriaans [2] (see Chapter 4) defines a system that uses a combination of PAC and Query Learning, suggesting that this is an appropriate model of syntax acquisition.

It should perhaps also be noted that the issue of having an oracle (in effect providing negative evidence) is under some debate from a psychological perspective (see Chapter 7). However, from a practical perspective, it seems unlikely that the Query Learning framework is a workable solution for large-scale grammar learning.

### 3.2.2.6 Compression as Learning

Inductive learning is commonly viewed as a process of compression, or optimisation [81]. In the context of natural-language grammar learning, compression is vital. The number of utterances that could need to be understood or generated is infinite, but there is only a finite space in which to represent the knowledge needed to perform the appropriate analysis. Wolff [143, 141, 142] has been one of the foremost proponents of considering natural language learning as compression from a psychological perspective, as well as a computational perspective.

It will be clear from the discussion above that I have used a compression approach in CLL. This is because it provides a simple approach to learning and a simple method to apply without forcing restrictions upon either the algorithm or the environment which would not be acceptable for natural-language syntax learning. Identification in the Limit, PAC Learning and Query Learning are not used as frameworks, because of both implausibility and restrictiveness. While compression is really an heuristic bias for a learning algorithm and not a framework, it does allow us to consider the learning algorithms with a degree of formality. However, future work into the better formalisation of learning models for natural language needs to be considered.

## 3.2.3 Stochastic Mechanisms

In the 1970s, speech recognition technology, i.e. the ability of a machine to reproduce as text an utterance by a person, was based on a two-stage process [69, 84]:

1. converting the sound wave of the utterance into a sequence of phonemes;



2. combining the sequences of phonemes to give sequences of words.

Each of these processes was completed using incomplete and inflexible rules. This led to rather inflexible speech recognisers, which had a limited vocabulary, could usually only recognise one person's voice and they could only handle simple subsets of language.

These problems were overcome to a large extent by the introduction, in the middle of the 1970s [69] of stochastic models, which allowed the recognisers to be trained with much larger vocabularies, over a greater subset of English, for many speakers.

Recently researchers in other areas of natural language processing, such as syntax and semantics, have realised that they are facing similar problems with grammars and rules that do not capture all the correct knowledge. This has led to a large number of attempts to use stochastic models in these settings in the hope that they will provide better results, as they did in speech recognition.

Statistical models are in general renowned in the natural-language-processing community for handling noisy or incomplete data well. As Bod describes [18], the models allow systems to make a *best guess* when faced with a situation in which a rule-based approach would simply “break”. However, even with stochastic models there is often a certain amount of brittleness. For example, if models are trained incorrectly or with incomplete data, sequences of words can be assigned zero probabilities when in fact they may be quite likely.

The second engineering motive, which is mentioned by Charniak [33], is that stochastic models are very amenable for learning. This is of course very important within the context of the present work. In principle, all that is required for learning is the collection of statistics from a corpus and application of these in the form of probabilities to the model. This is, however, somewhat of a simplification, as will be shown. The fundamental problem here is that it is necessary to define the model and then apply the statistics. In parsing and grammar learning, this model is the grammar, or in the case of CLL the lexicon. In practice, it will often be necessary or useful to also learn at least part of the underlying model. As has already been discussed, for the model presented here, the aim is to learn the mappings between words and categories as well as the probabilities of those mappings.

Importantly though, there are efficient algorithms for training most stochastic models (i.e. setting the probabilities). Also, there are usually efficient algorithms for using the models, for example for selecting the most likely structure given an utterance. This has led to improvements in parsing efficiency, as it is possible to avoid the structural ambiguity problem [18, 84] and even category disambiguation [53].

Again it is worth noting that there is strong evidence that humans use frequency-based methods in analysing language and so stochastic approaches can be considered to be psychologically plausible. More than this, it suggests the artefact of natural language can be considered from a stochastic perspective. Chapter 7 contains a more indepth discussion of the psychological plausibility of using stochastic methods.

In conclusion, the use of stochastic models provides ways of implementing robust and relatively efficient systems, as well as being potentially psychologically plausible. It was mainly motivated by the success of such models in speech recognition, but work has shown it to be useful in modelling language at a higher level [18, 84, 44].



### 3.3 Computational Language Learning Conclusions

In this chapter a computational model for natural-language syntax-learning has been presented. In this section, I will aim to summarise the above discussion firstly to make the problem that is to be solved clear and secondly to discuss the type of system that will be used to solve this problem.

#### 3.3.1 The Problem

The natural-language syntax-learning problem can be characterised in terms of the input and output of the learner. The input is a set of examples where each example is a sentence with either no annotation, or a very small amount of annotation. The sentences will be considered as lists of words and will commonly be those generated as written text rather than spoken text.

The aim of the system is to build natural-language grammars. In particular, having settled upon Categorical Grammar as the formalism for representing the learned syntactic information in Chapter 2, the output of the system must at least be a CG lexicon.

Hence, the problem of building a natural-language syntax-learning system for our setting can be described as building a system that takes a set of the examples as input (dealing with each in turn) and produces a CG lexicon which can be used to assign the correct CG categories to the words in the examples. This is the same as saying that the lexicon can be used to build the correct parses for the examples.

#### 3.3.2 The Type of System

The type of system has been characterised above as a search engine, which, given the examples, searches the space of possible grammars. The aim is to find the best grammar in the space. Because the search space is large, a set of constraints must be used both to reduce the size of the search space and bias the learner towards the areas that will return most productive grammars.

The search engine is constrained using linguistic knowledge provided by the CG formalism, the current state of syntactic knowledge and some sort of bootstrapping knowledge about the kind of lexicons that can be built.

There are also some learning principles used to constrain the search engine, in particular to determine what the “best” grammars in the space are. Compression and maximum likelihood methods are used in this model.

Finally, there are also some cognitive constraints on the search engine that may be considered. These constrain the amount of processing performed by the learner on the problem, to ensure that the learner is efficient. The cognitive constraints also seek to include some of the psycholinguistic constraints on syntactic processing, e.g. dealing with words and sentences incrementally.

From the discussion as to the type of algorithms and frameworks that might be used for implementing this algorithm, the conclusion is that a combination of symbolic and stochastic methods is to be used within a compression-based learning setting.

In this chapter, the computational learning model has been defined and the type of algorithms that will be used in building an implementation have been discussed. The next chapter discusses alternative approaches to solving these kinds of problems, which have been pursued by others.



## Chapter 4

# Computational Syntax-Learning Systems

In the previous chapters I have gone into some detail describing the type of problem that is to be solved within the context of natural-language syntax-learning. The problem has been set in its context and a model for solving it has been suggested. This model has been implemented as the Category Label Learner CLL. The implementation is discussed in detail in the next chapter. In this chapter the alternative approaches that have been pursued by others are discussed and compared with the model pursued in this thesis. The aim is to show that the model is both unique in its setting and also unique in its approach to handling the learning task.

The computational learning model described in the last chapter is similar to both the training section of part-of-speech tagging systems and grammar learning systems. The system builds a mapping between a set of tags and a set of words making it similar to part-of-speech tagging. In fact, it would be entirely possible to extract a lexicon from a part-of-speech tagged corpus. However, there are two crucial differences. Firstly, part-of-speech tagging is essentially a disambiguation task. The systems are always, in effect, supplied with a lexicon which contains all possible parts-of-speech for each word. In contrast, CLL has, at most, a partial version of this lexicon. Secondly, the CG tags are of a complex type, containing a lot of syntactic information. In fact, CLL learns more than just a mapping. In the process it learns the syntactic function-argument structure of the language (which is the same as the semantic function-argument structure in CG), as this is the defined meaning of the CG categories. This is in contrast to part-of-speech tagging, when words are just grouped together under the same labels and the only meaning of the annotation is based on the meaning a human applies to the set of tags. This contrast indicates the similarity between CLL and a grammar learning system.

There is a sense in which the approach followed with CLL can be seen as a conservative extension of part-of-speech tagging. As Abney [1] suggests, part-of-speech tagging has been used to show that acceptable results can be achieved on NLP problems without having to handle the entire NL understanding problem. The extension we are considering, somewhere between the training stage of part-of-speech tagging and full grammar learning is the ideal next stage.

With this in mind, the review of systems below is split into two sections: part-of-speech tagging systems (Section 4.1) and syntax-learning systems (Section 4.2).

## 4.1 Part-of-Speech Tagging

The production of a part-of-speech tagging system has two stages. Firstly, a model representing how parts of speech can be assigned to words in a sentence needs to be built (e.g. a set of rules, or probabilities). Secondly, that model is applied to unseen sentences to assign the appropriate parts-of-speech to the words. In this thesis, the aim is to build lexical resources, so the extraction of linguistic knowledge in the first stage of building a part-of-speech tagger is of most interest (although the second stage is useful for evaluating the quality of the model built). The review below will concentrate on the approach to building the model with reference to the results that can be achieved with the model to indicate how good it is.

The methods used for part-of-speech-tagger training can be categorised into two types, supervised and unsupervised. Within this categorisation there are also those that use (mostly) symbolic methods (e.g. rules) to tag the examples in the corpus, and those that use (mostly) stochastic methods (e.g. Markov models). Some of the approaches combine the two different types of method to some degree, which is what I have aimed to do with CLL. Hence, comparison at this point will be interesting. Obviously, the most interesting systems will be those that are unsupervised and use a mixture of symbolic and stochastic methods, as these will allow the closest comparison with the model I have defined. It will be seen that very few systems with that combination of features exist. In Section 4.1.1 the supervised approaches are presented and discussed and in Section 4.1.2 the unsupervised approaches are presented and discussed.

It has already been mentioned that the approach implemented in CLL is beyond part-of-speech tagging, but perhaps somewhat short of full grammar learning. A similar idea has been pursued in the *supertagging* work of Joshi and Srinivas [72], although they use a supervised approach. This work is described in Section 4.1.3.

### 4.1.1 Supervised Part-of-Speech Tagging

While supervised approaches are less interesting in the context of the model described in the previous chapter, it is still the case that it is worth reviewing them. Firstly, they often provide insights into the learning problem. Much of what is contained in this section is directly applicable to unsupervised learning. Secondly, they are commonly the basis for unsupervised approaches. Finally, there are far fewer unsupervised systems, so it is important to use as much of the knowledge available from supervised systems as possible.

Early part-of-speech tagging models, e.g. the TAGGIT and CLAWS systems (discussed by Abney [1]), consisted of sets of hand-built rules that were used to determine the correct tag for words in examples. The context of the word was used to determine which of the possible tags the word could take was the correct tag for the word in a particular context. There is no learning involved in this process and so this is of less interest, except to note that these hand-built systems performed much less well than systems that allowed an element of training, both with respect to accuracy and robustness. The TAGGIT system achieved a



77% accuracy on the Brown corpus and had to be hand corrected. This accuracy value is much lower than the systems below, which include a learning element. Such results provide a further indication that it will be useful to develop more learning systems for NLP tasks. However, as Brill notes [26], the advantage of rule-based systems is that the linguistic knowledge that they contain is perspicuous, unlike many of the non-symbolic approaches, which hide the linguistic knowledge in tables of probabilities or weightings. Ideally, a flexible and robust rule-based system should be built, probably using a symbolic learning system. There have been some effective recent approaches to this.

Brill [23, 26, 27] has developed the Transformation-Based Error-Driven Learning model, which is shown diagrammatically in Figure 4.1 (taken from Brill e.g. [26]). The setting may initially look to be unsupervised, as the input is *unannotated text*, however, this is not the case. The *initial state* is built in a variety of ways. The first approach [23] was to assign each word in an example its most common part-of-speech, which is determined from the (tagged) training corpus (unknown words were tagged using a variety of heuristics). Other approaches have included using the output of a stochastic *n*-gram part-of-speech tagger and also assigning all words a noun part-of-speech. Whatever process is used to give the initial state, it leaves an *annotated text*. This state is then compared with the *truth*, which in this case is the manually tagged version of the text which was initially presented to the learner. Hence, the system is strongly supervised. The comparison with the truth is used to build an ordered list of transformations, which make the annotation closer to the truth. A transformation contains two parts, which Brill [23, 26] calls the *rewrite rule* and the *triggering environment*. The rewrite rule is simply:

*Change the tag from a to tag b*

where **a** and **b** are variables over the set of part-of-speech tags. Initially [23] there were thirteen templates for the triggering environment, e.g.

*The preceding word is tagged as a x*

where **x** ranges over the set of part-of-speech tags. This set of templates was extended to capture more lexical relationships [25, 26]. A further ten templates were added that referred to words instead of tags, e.g.

*The preceding word is a w*

where **w** is a variable ranging over the entire set of words. An example of a complete transformation [26] would be:

Change the tag from *modal* to *noun* when:  
The preceding word is tagged as a *determiner*.

This would correctly change the sequence:

The/determiner can/*modal* rusted/verb.

into:

The/determiner can/*noun* rusted/verb.

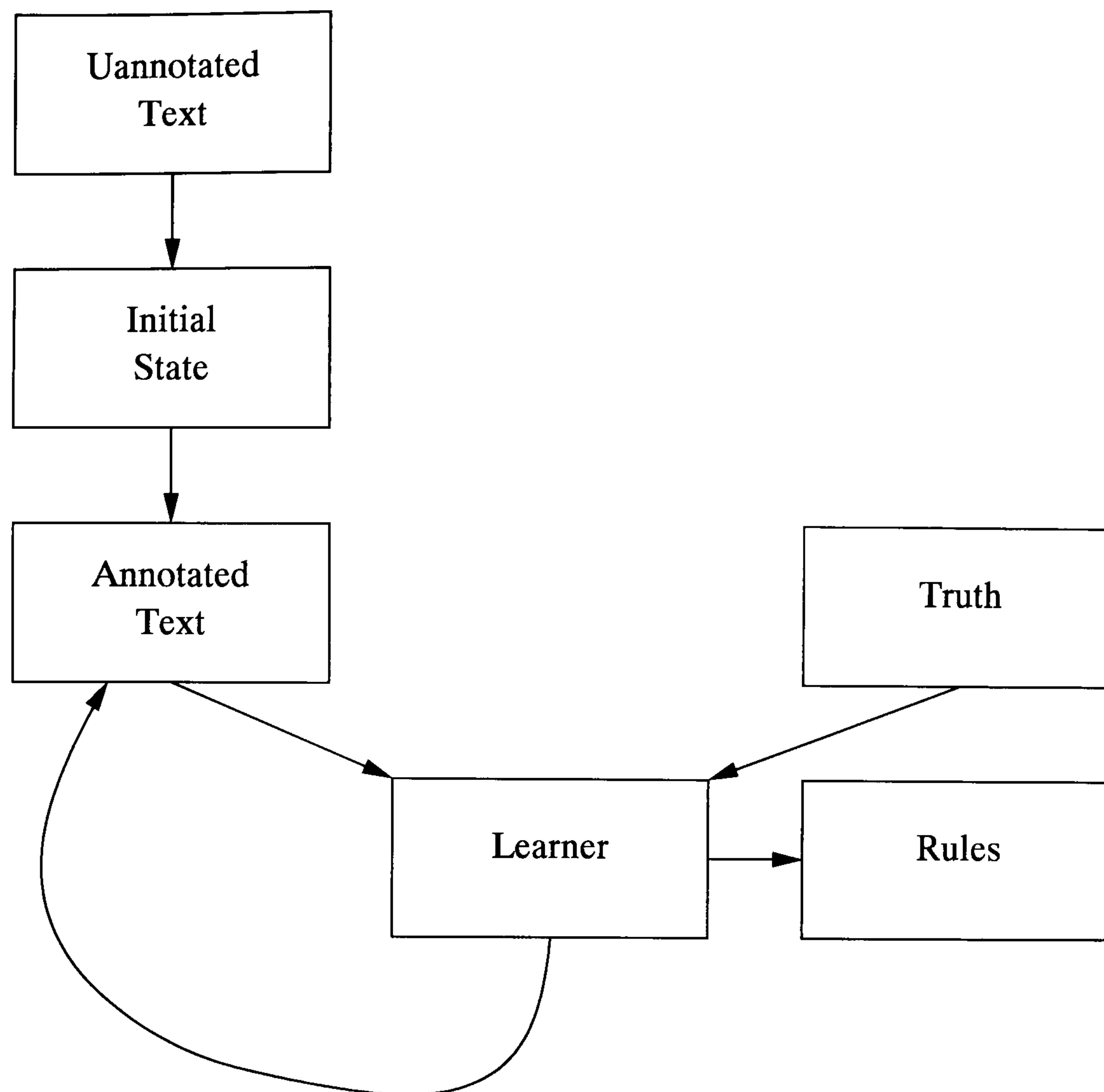


Figure 4.1: A graphical representation of the Transformation-Based Error-Driven Learning model

An ordered list of such transformations is built up incrementally. A transformation is added to the bottom of the list if it is currently the *best* transformation, by which is meant some notion of the most improvement to the annotation of the corpus. The transformation is then applied to the corpus and the process is repeated until no transformation can be found that improves the corpus. The notion of the best transformation, is simply the transformation that most reduces the number of errors in the corpus when compared against the truth. To find this transformation, a data-driven search of all possible transformations is applied.

In later work [25, 26], as well as lexicalising the triggering templates, the training element of the system was also extended to learn transformation rules for unknown words and to return the  $k$ -best tags rather than simply the best tag for a word.

Brill [26] reports results when this part-of-speech tagging system was trained on 600,000 words from the Penn Treebank, using the most likely tag from the training corpus to define the initial state (in this case the test set was used incorporated in the lexicon, but used in no other way, so that unknown words did not have to be dealt with). A total of 447 transformations were learned. When the learned part-of-speech tagger was applied to 150,000 word test set, a tagging accuracy of 97.2% was achieved. If the lexical triggering templates are not used, then the accuracy drops to 97.0%. (For a more detailed presentation of results, including earlier results and also results with the  $k$ -best tags system, see Brill [25, 26]).

This system has some similarities with the one proposed for category learning in Chap-



ter 3. The system uses symbolic methods along with a simple notion of best to determine the correct addition to the learned knowledge. The transformation templates could be considered to be similar syntactic constraints to the CG categories and application rules in CLL.

However, there are important differences. Apart from the obvious restriction of learning only parts-of-speech, rather than CG categories, which has been discussed above (although see Section 4.2 for the application of the techniques to syntax learning), the system is also strongly supervised, which, as has been said, is hard to justify practically (as well as psychologically). While this may be a practical engineering approach in a simple part-of-speech tagging context, where a manually tagged corpus already exists, such a corpus is not available tagged with CG tags. The use of the lexicon, as discussed above, also reduces the task simply to disambiguation, modifying a mapping rather than building it. In this case, as the lexicon is calculated from the full (training and test) corpus a complete initial mapping is also given to the learner.

All these differences indicate a system solving a much simpler problem than CLL. Hence, the results will not really be comparable. However, the techniques used in the system, as noted above, bear some similarity with CLL and this will become of more interest as Brill has applied the Transformation-Based Error-Driven Learning model to syntax learning [23, 24, 22, 26, 111] (see Section 4.2) and also to *weakly supervised* or *unsupervised* part-of-speech tagging [27].

Cussens [48, 49] presents an alternative supervised system that uses an Inductive Logic Programming (ILP) [93] algorithm, Progol [92], to learn rules to eliminate categories in a somewhat similar way to Brill's unsupervised approach [27]. Initially a lexicon is built from the part-of-speech-tagged corpus (the WSJ section of the Penn Treebank) and the system described is only to be used on corpora with all words in this lexicon. The lexicon contains the mapping between words and all the categories they are assigned in the corpus, along with the frequencies of those assignments. Given this lexicon, the system is again reduced to doing only disambiguation. This is achieved with a combination of symbolic and stochastic methods.

Following the lexicon building, a set of elimination rules are learned using the Progol algorithm (with some caching extensions for efficiency). The elimination rules are learned for each tag, i.e. each ambiguous tag has a set of elimination rules attached to it. To learn these the Progol algorithm is given a set of positive and negative examples and a partial grammar.

The positive and negative examples are generated from the training corpus using the part-of-speech annotation. Sentences are taken one at a time and each word that can take more than one tag is used to generate one or more positive examples and one negative example. A positive example is the elimination of one of the tags which is not the tag used in the sentence for this word. The negative example is the removal of the part-of-speech tag which is the tag used in the sentence for this word. Generally, this presented far too many examples and they were reduced in various ways to give at most 6000 examples per tag. The examples contain the entire context to the left and the right of the tag in question, and the tag to be removed. The context is taken from the correctly tagged corpus.

Cussens' system also uses a partial grammar as background knowledge. The learner



uses this syntactic knowledge to constrain the learning process. This is not dissimilar to CLL, which uses the syntactic knowledge of the CG application rules and the categories to constrain the learning process. The grammar is simple and, as Cussens notes [48], is rather over-general, but it is only used to improve the tag elimination rules.

Where the elimination rules did not completely disambiguate the tags, a simple stochastic approach of choosing the most frequent tag of those that remain was used. Hence, there is a mixture of symbolic and stochastic approaches.

Progol was used on the examples extracted and built a theory with 885 clauses (in effect 885 rules) to be used for tagging. The learned tagger was then applied to a test set of 5000 examples from the Penn Treebank, which consisted of 110,716 tokens, although only examples starting with a capital letter, ending in a full stop and containing words in the lexicon could be tagged. Overall the accuracy of the system is 96.4%. Interestingly, Cussens also provides a value just using the most probable tag for a word, which is 94.1%.

In many ways this system is interesting. The mix of symbolic and stochastic approaches is similar in some ways to the approach in CLL, where the symbolic constraints are similarly given precedence. The use of simple frequency information seems reasonable from a psycholinguistic perspective. The system is also the first to have used some syntactic knowledge, by applying a partial grammar to the task. Although this may be somewhat psychologically implausible for a system learning part-of-speech tagging, it does have some similarities with the approach used in CLL for a more complicated task.

The system is, however, very supervised. A lexicon is used (including frequency information), which is extracted from the tagged corpus and the examples given to Progol are extracted using information on the correct tag from the tagged corpus. Also the system uses positive and negative examples. It may be argued that these are generated by the learner from the environment, however this is only possible because the environment contains the tagging information. Hence, the system uses a lot of information, which is not available to CLL.

The stochastic approaches to part-of-speech tagging have probably been more popular than the symbolic approaches. They have certainly been particularly effective. The most commonly used model is the tri-gram Markov Model approach. Initially the theory behind this model will be discussed with respect to probabilistic language models and then the model will be applied specifically to part-of-speech tagging.

#### 4.1.1.1 Markov Chains

A Markov chain is simply a deterministic finite-state automata with probabilities attached to the transitions. Figure 4.2 shows an example of such a chain given by Charniak [33]. Markov chains give probabilities for the strings of symbols they generate or accept.

As finite state automata are machine representations of type 3 (regular) grammars, this shows that Markov chains represent stochastic regular grammars. While this may be a suitable generative power for speech recognition tasks it is undoubtedly limiting when considering natural language syntax, which is in general considered to require context-free or greater generative power. However, it has proved a useful approximation, particularly in part-of-speech tagging.



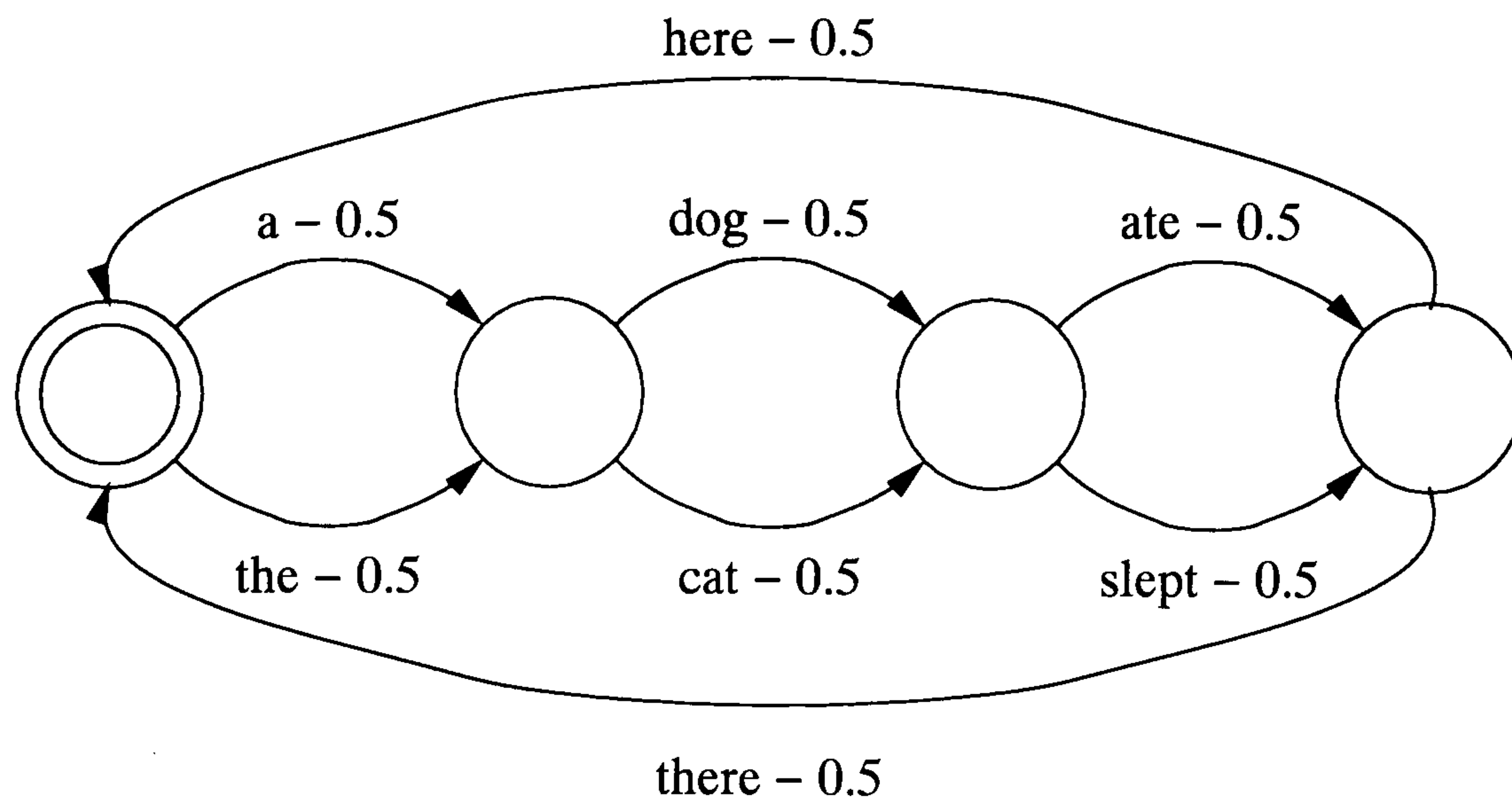


Figure 4.2: An example of a Markov Chain

#### 4.1.1.2 *N*-gram Models

*N*-gram models are perhaps the simplest stochastic language models. They are based on the assumption that the probability of a word in an utterance is based solely on the previous  $n - 1$  words. More formally:

$$\begin{aligned}
 P(w_i) &= P(w_i | w_1, w_2, \dots, w_{i-1}) \\
 &= P(w_i | w_{i-(n-1)}, w_{i-(n-2)}, \dots, w_{i-2}, w_{i-1})
 \end{aligned}$$

This is a very strong assertion, and knowledge of the structure of natural language suggests it is incorrect. However, it is a simple model both to build and use. It has proved useful in a number of contexts, from machine translation [28] to machine learning of musical regularities [102].

Estimating the probabilities can be expensive, as it requires extensive collection from a training sample, although this process need only be completed once. Also, the greater the size of  $N$  the more sparse the data will be and the larger the number of  $N$ -grams that will have to be estimated. Therefore a common value for  $N$  is three, yielding a *tri-gram* model, which allows for some context, without the number of  $N$ -grams becoming unmanageable. In such a case it is necessary to collect data of frequencies of sequences of three words. Following Charniak, [33], the estimated probability of the occurrence of a word  $w_i$  given the previous two words, where frequencies are denoted by  $C$  is then given by:

$$P_e(w_i | w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}$$

In other words, the number of times the tri-gram appears divided by the total number of times the context appears.

*N*-gram models can be represented concisely as Markov chains. Figure 4.3 shows an example adapted from Charniak [33] where the language is simply sequences of “a”s and “b”s. Each state represents what the previous two inputs were.

Such *N*-gram models suffer from being trained with sparse data. In other words, if a trigram is not in the training sample then it is assigned the probability zero. Hence, the

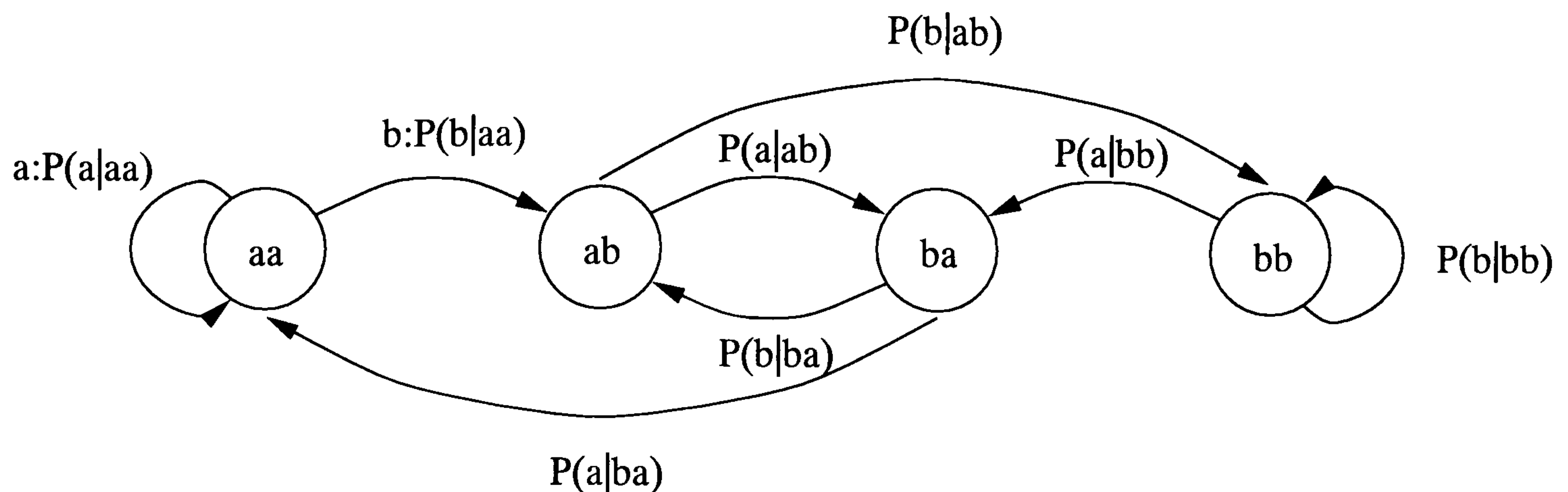


Figure 4.3: An example tri-gram model

model will not be able to handle any text with that tri-gram in it. It is surprisingly likely that new texts will have a large number of unseen trigrams. To handle this the probabilities of words are “smoothed” with the probabilities of bi-grams and uni-grams i.e.

$$P(w_n|w_{n-2}, w_{n-1}) = \lambda_1 P_e(w_n) + \lambda_2 P_e(w_n|w_{n-1}) + \lambda_3 P_e(w_n|w_{n-2}, w_{n-1})$$

Where  $\lambda_1, \lambda_2$  and  $\lambda_3$  are non-negative constants to scale the influence of uni-gram, bi-gram and tri-gram probabilities respectively. To maintain the integrity of the probability calculation,  $\lambda_1 + \lambda_2 + \lambda_3 = 1$ .

However, as there are may be several ways of generating the same string of words giving different probabilities, it is not possible to use Markov chains as a model, as they are deterministic (i.e. there is only one way of generating the string). In the next section, Hidden Markov Models are presented as a more flexible model that can handle this non-determinism.

#### 4.1.1.3 Hidden Markov Models (HMMs)

Hidden Markov Models (HMMs) are essentially non-deterministic finite-state automata with probabilistic enrichment. As already noted, this is to allow more flexible language models for including more than one way of calculating the probability of a sequence of words. HMMs are particularly popular in speech recognition [69, 9] and part-of-speech tagging [33].

To begin with I define HMMs which will be followed by a description of how HMMs can be used. HMMs are discussed in some detail below as the approaches used with such relatively simple stochastic models can usually be modified for use with other stochastic models of language.

**4.1.1.3.1 The Definition of HMMs** There appear to be two distinct definitions of HMMs. The first, taking Markov chains and making them non-deterministic, is presented by Charniak [33]. The second, defining HMMs with two distinct stochastic processes, is presented by Rabiner and Juang [103]. Here I will follow Charniak’s presentation [33].

Charniak [33] defines Hidden Markov models with the four-tuple  $\langle s^1, S, W, E \rangle$  where  $S$  is the set of states of the machine,  $s^1$  is the initial state of the model,  $W$  is the set of output symbols and  $E$  is the set of transitions, or edges. A transition is defined as a four-tuple  $\langle s^i, s^j, w^k, p \rangle$ , which defines a transition as starting in state  $s^i$  and going to state  $s^j$



(where  $s^i, s^j \in S$ ). The output symbol of the transition is  $w^k$ , where  $w^k \in W$  and  $p$  is the probability that the transition is taken. A transition may be written  $s^i \xrightarrow{w^k} s^j$  with the probability  $p$ . The only restriction on transitions is that no redundant transitions are allowed, in other words given two transitions  $s^a \xrightarrow{w^x} s^b$  and  $s^c \xrightarrow{w^y} s^d$  with arbitrary probabilities, then  $\neg(s^a = s^c \wedge s^b = s^d \wedge w^x = w^y)$ . Any such transitions can easily be combined into one transition by adding the probabilities. To find the probability of a sequence of transitions the probabilities of all the separate transitions are multiplied. Note that the probabilities of the transitions from a given state must sum to one. This ensures that the probabilities of all the strings an HMM can generate of a given length will sum to one.

As mentioned above, HMMs are essentially a non-deterministic finite-state automata with probabilities attached to the transitions. Hence, given a particular output sequence it may not be possible to say what sequence of states and transitions were used to generate that sequence. Hence, the notion of a *hidden* element in the model. Such models can be represented graphically in the same way as Markov chains. Figure 4.4 is an example of an HMM for some language with the vocabulary  $\{a, b\}$ .

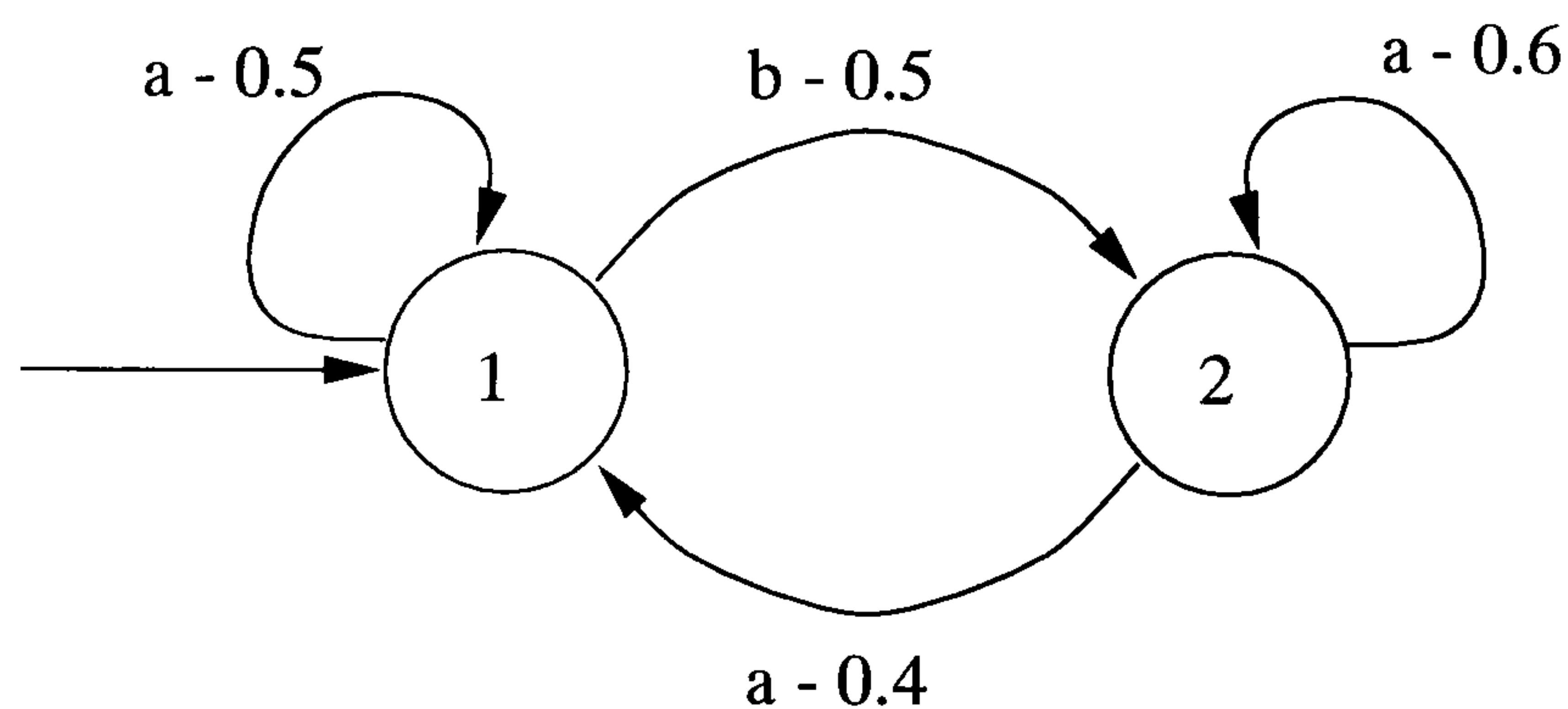


Figure 4.4: An example Of a Hidden Markov model

Given the string *baa* it is easy to see that there are three ways to generate this string from the HMM in Figure 4.4. Firstly, the sequence of transitions  $1 \xrightarrow{b} 2, 2 \xrightarrow{a} 2, 2 \xrightarrow{a} 2$  can be followed, which has the probability  $0.5 \times 0.4 \times 0.4 = 0.08$ . Secondly, the sequence of transitions  $1 \xrightarrow{b} 2, 2 \xrightarrow{a} 1, 1 \xrightarrow{a} 1$  can be followed, which has the probability  $0.5 \times 0.6 \times 0.4 = 0.12$ . Finally, the sequence of transitions  $1 \xrightarrow{b} 2, 2 \xrightarrow{a} 1, 1 \xrightarrow{a} 1$  can be followed, which has the probability  $0.5 \times 0.4 \times 0.5 = 0.1$ .

#### 4.1.1.4 Problems and Solutions with HMMs

From the above definition it is clear that an HMM may be useful for stochastic language modelling, however there are several questions that must be answered before these models can be constructed or used. In this section we consider these questions and aim to outline some of the solutions from the literature.

**4.1.1.4.1 Three or Four Problems with Markov Models?** Both Rabiner and Juang [103] and Charniak [33] suggest that there are three problems for which it must be possible to find answers.

1. Calculating the probability of an observation sequence.

2. Finding the most probable state sequence for an observation sequence.
3. Calculating the probability distributions for the model.

The first of these allows the comparison of models for given sequences. The second is an attempt to discover information on the hidden part of the model. The last problem is to *train* the model, so that the probabilities assigned give the best model for the particular part of language we are interested in.

Whilst all these problems must be solved, Stolcke and Omohundro [120] suggest a further problem, which is to produce the structure of the HMM in the first place, i.e. the number of states and their connectivity (note, if two states are not connected for a particular observation symbol, it is equivalent to there being a probability of zero for that event).

The solutions to these problems will be investigated in some detail because they have been used extensively in part-of-speech tagging and these solutions commonly form the basis for solving the same problems when using other stochastic language models.

**4.1.1.4.2 Finding the Probability of an Output Sequence** To calculate the probability of an output sequence, it is necessary to sum the probabilities of all possible state sequences that cause this output sequence. However, to calculate all these separately would give a time complexity that was exponential with respect to the length of the sequence. Fortunately there is an algorithm that is linear with respect to the length of the sequence [103, 33].

The algorithm involves calculating the *forward* probabilities  $\alpha_i(t)$  (so called because the probabilities are calculated by moving through the output sequence from beginning to end), the probability of producing the output sequence  $w_{1,t-1}$  with the ending state being  $s^i$ , i.e.

$$\alpha_i(t) \stackrel{\text{def}}{=} P(w_{1,t-1}, S_t = s^i)$$

It is clear that:

$$P(w_{1,t-1}) = \sum_{i=1}^{\sigma} \alpha_i(t)$$

Where  $\sigma$  is the number of states. So all that needs to be shown is that  $\alpha_i(t)$  can be calculated efficiently. Charniak [33] gives the standard linear recursive algorithm as follows:

**Base case:**

$$\alpha_i(1) = \begin{cases} 1.0 & \text{if } i = 1 \\ 0.0 & \text{otherwise} \end{cases}$$

**Recursive case:**

$$\begin{aligned} \alpha_j(t+1) &= P(w_{1,t}, S_{t+1} = s^j) \\ &= \sum_{i=1}^{\sigma} P(w_{1,t}, S_t = s^i, S_{t+1} = s^j) \\ &= \sum_{i=1}^{\sigma} P(w_{1,t-1}, S_t = s^i) P(w_t, S_{t+1} = s^j | w_{1,t-1}, S_t = s^i) \\ &= \sum_{i=1}^{\sigma} P(w_{1,t-1}, S_t = s^i) P(w_t, S_{t+1} = s^j | S_t = s^i) \\ &= \sum_{i=1}^{\sigma} \alpha_i(t) P(s^i \xrightarrow{w_t} s^j) \end{aligned}$$



One could calculate the same probabilities in reverse, starting at the end of the string and working to the beginning. Hence, one can define the *backward probability* that a given substring length  $t$  ends in state  $s^i$ :

$$\beta_i(t) \stackrel{\text{def}}{=} P(w_{t,n} | S_t = s^i)$$

which will be shown to be useful in training the probabilities of HMMs.

Using this definition the probability of the string *ba* using the HMM in Figure 4.4 can be calculated.

$$\begin{aligned} \alpha_1(3) &= (\alpha_1(2) \times 0.5) + (\alpha_2(2) \times 0.4) \\ &= ((\alpha_1(1) \times 0) + (\alpha_2(1) \times 0) \times 0.5) + ((\alpha_1(1) \times 0.5) + (\alpha_2(1) \times 0) \times 0.4) \\ &= 0 + (((1 \times 0.5) + 0) \times 0.4) \\ &= 0.2 \end{aligned}$$

$$\begin{aligned} \alpha_2(3) &= (\alpha_1(2) \times 0) + (\alpha_2(2) \times 0.6) \\ &= 0 + (((\alpha_1(1) \times 0.5) + (\alpha_2(1) \times 0)) \times 0.6) \\ &= ((1 \times 0.5) + 0) \times 0.6 \\ &= 0.3 \end{aligned}$$

Hence:

$$\begin{aligned} P(ba) &= 0.3 + 0.2 \\ &= 0.5 \end{aligned}$$

Note that the probabilities of all two letter sequences will sum to one.

**4.1.1.4.3 Getting the Most Probable Hidden State Sequence** A standard method for discovering the most probable hidden-state sequence is known as the *Viterbi algorithm*. Both Charniak [33] and Rabiner and Juang [103] describe it.

It is clearly too inefficient to calculate the most probable path for a sequence by calculating the probabilities of all the possible paths. It is also unnecessary, because, when generating the state sequence, it is possible to prune many paths which will never be part of a most probable sequence. To see this consider generating the sequence *baabaa* using the HMM in Figure 4.4. There are two ways of generating *baa* and ending up in state 1:

1.  $1 \xrightarrow{b} 2, 2 \xrightarrow{a} 2, 2 \xrightarrow{a} 1$  with probability 0.12;
2.  $1 \xrightarrow{b} 2, 2 \xrightarrow{a} 1, 1 \xrightarrow{a} 1$  with probability 0.1.

The second of these will never be a part of a most probable sequence instead of the first, because exactly the same sequences of transitions can come before and after both these sequences and so the first will always make the probability higher than the second.

It is, however, possible that two state sequences ending in different states for the same output sequence could be part of the most probable sequence whatever their respective

probabilities. For example, the highest probability for the subsequence *baa* using the HMM in Figure 4.4 is state sequence:  $1 \xrightarrow{b} 2, 2 \xrightarrow{a} 2, 2 \xrightarrow{a} 2$ , with the probability 0.18. However, this sequence ensures a probability of 0 for *baabaa* if this is used as the first three transitions in the sequence, as it is not possible to then generate a *b*.

These intuitions suggest the Viterbi algorithm. Charniak [33] recursively defines a function  $\sigma_i(t)$  which returns the most probable state sequence length,  $t$ , that ends in state  $s^i$ , for a given output sequence length  $t - 1$ .

$$\begin{aligned}\sigma_i(1) &= s^i \\ \sigma_i(t+1) &= \sigma_j(t) \circ s^i\end{aligned}$$

where  $j$  is selected as follows:

$$j = \operatorname{argmax}_k P(\sigma_k(t))P(s^k \xrightarrow{w_i} s^i)$$

where  $k$  ranges over the indices of the set of states. Hence, we have the most probable state sequences to all states for  $t$ .

Hence, at each stage in the search tree it is possible to cut out large parts of the tree below that will never lead to the most probable solution. It in fact makes the algorithm linear in the length of the output sequence with respect to time complexity.

For example, calculating the most probable state sequence given the sequence *baabaa* involves keeping track of sequences ending in state 1 and state 2. Table 4.1 shows that the most probable sequence of states is  $\langle 1, 2, 2, 1, 2, 2, 2 \rangle$ , with a probability 0.0216.

Length	Output	States	Probability
1	$\epsilon$	$\langle 1 \rangle$	1.0
		$\langle 2 \rangle$	0.0
2	b	$\langle 1, 1 \rangle$	0.0
		$\langle 1, 2 \rangle$	0.5
3	ba	$\langle 1, 2, 1 \rangle$	0.2
		$\langle 1, 2, 2 \rangle$	0.3
4	baa	$\langle 1, 2, 2, 1 \rangle$	0.12
		$\langle 1, 2, 2, 2 \rangle$	0.18
5	baab	$\langle 1, 2, 2, 1, 1 \rangle$	0.0
		$\langle 1, 2, 2, 1, 2 \rangle$	0.06
6	baaba	$\langle 1, 2, 2, 1, 2, 1 \rangle$	0.024
		$\langle 1, 2, 2, 1, 2, 2 \rangle$	0.036
7	baabaa	$\langle 1, 2, 2, 1, 2, 2, 1 \rangle$	0.0144
		$\langle 1, 2, 2, 1, 2, 2, 2 \rangle$	0.0216

Table 4.1: Example of the calculation of the most probable sequence of states

**4.1.1.4.4 Training HMMs** The algorithm for calculating the probabilities of the transitions in an HMM is the *Baum-Welch* or *forward-backward* algorithm, which is an example of the Expectation Maximisation (EM) algorithm [52] (here Charniak's presentation of the algorithm is followed [33]). The aim is to use a training sequence for which the probability



```

Old_Sample_Probability = 0.0
Guess HMM Transition Probabilities
New_Sample_Probability = Re-estimate_Transition_Probabilities
Loop until (Old_Sample_Probability)  $\approx$  New_Sample_Probability
    Old_Sample_Probability = New_Sample_Probability
    New_Sample_Probability = Re-estimate_Transition_Probabilities

```

Figure 4.5: The HMM training algorithm

assigned by the HMM is to be maximised.

Given a simple Markov chain model, then this would be easy. Using the training sequence, it would be a matter of counting the number of times a transition was used  $C(s^i \xrightarrow{w^k} s^j)$  and then dividing that by the total number of transitions taken from that state i.e.

$$P(s^i \xrightarrow{w^k} s^j) = \frac{C(s^i \xrightarrow{w^k} s^j)}{\sum_{l=1, m=1}^{\sigma, \omega} C(s^i \xrightarrow{w^m} s^l)}$$

However, with an HMM, this is more complicated, as there are many paths to produce most output sequences. Charniak [33] describes two insights for solving this problem. The first is that when there is a choice between transitions leading to different paths for the same output sequence, then it is assumed that *all* are considered, but each is multiplied by the probability of the path to scale them. This is written formally as:

$$\begin{aligned} C_e(s^i \xrightarrow{w^k} s^j) &= \sum_{s_{1,n+1}} P(s_{1,n+1} | w_{1,n}) \eta(s^i \xrightarrow{w^k} s^j, s_{1,n}, w_{1,n}) \\ &= \frac{1}{P(w_{1,n})} \sum_{s_{1,n+1}} P(s_{1,n+1}, w_{1,n}) \eta(s^i \xrightarrow{w^k} s^j, s_{1,n}, w_{1,n}) \end{aligned}$$

Here the function  $C_e(x)$  returns the estimated number of times the transition  $x$  will be taken and  $\eta(s^i \xrightarrow{w^k} s^j, s_{1,n}, w_{1,n})$  is the number of times  $s^i \xrightarrow{w^k} s^j$  occurs in the state sequence  $s_{1,n}$  with the output  $w_{1,n}$ .

The second insight is that it is necessary to know the probability of the path to perform this calculation, so it seems it is necessary to know the probabilities of the arcs before calculating them. This is solved by guessing an initial set of probabilities and refining these using the hill-climbing algorithm in Figure 4.5 (taken from Charniak [33]). This maximises the probability of the training examples.

However, it is still necessary to find an efficient way of re-estimating the probabilities. Charniak [33] shows it is possible to express the function  $C$  in terms of the forward and backward probabilities defined previously (see Charniak [33] for the details of the derivation).

$$C(s^i \xrightarrow{w^k} s^j) = \frac{1}{P(w_{1,n})} \sum_{t=1}^n \alpha_i(t) P(s^i \xrightarrow{w^k} s^j) \beta_j(t+1)$$

Using this definition, it is possible to re-estimate the probabilities for the transitions linearly.

As an example of the application of this algorithm, suppose that the initial model with guessed probabilities is the HMM shown in Figure 4.4. The first two iterations of the algo-

Transition	$P_1$	$C_1$	$P_2$	$C_2$	$P_3$
$1 \xrightarrow{a} 1$	0.5	3.51	0.57	4.16	0.63
$1 \xrightarrow{b} 2$	0.5	2.66	0.43	2.46	0.37
$2 \xrightarrow{a} 2$	0.6	2.31	0.62	2.27	0.62
$2 \xrightarrow{a} 1$	0.4	1.39	0.38	1.41	0.38

Table 4.2: Examples of the figure calculated by the forward-backward algorithm

rithm are given in Table 4.2 given a training sequence *abaaaab*. In this particular example, the probabilities of transitions out of state two settle quickly.

This training model is very useful and can even be applied in an unsupervised way [78]. However, Charniak [33] identifies the obvious limitations, which are that the hill-climbing iterative algorithm will lead to the discovery of a local maximum model rather than the best model and the model has a tendency to over-fit to the training data, since it is a maximum likelihood (ML) technique.

**4.1.1.4.5 Finding the Initial Stochastic Model** It has already been discussed that there is an extra problem with stochastic models. It is necessary to define an initial state and structure for any stochastic model, which is to be trained. It is common that the model is pre-defined by the user, however automatic induction of this model has been investigated by Stolcke and Omohundro [120]. The approach they develop is intended to apply to all stochastic models and consists of two stages.

**Data incorporation** A model  $M_0$  is built, which includes each data point from the example set  $X$  individually. In this construction, the aim is to maximise  $P(X|M)$ .

**Structure Merging** Build a sequence of models gradually maximising  $P(M|X)$ , so that each new model is an improvement.

This is clearly a general approach, however, examples of how it can be applied to both HMMs and P-CFGs are provided by Stolcke and Omohundro [120]. It should be mentioned that these approaches are very similar to the grammar induction methods of Angluin [5] and Sakakibara [105, 106], but here structure merging is performed to maximise probability. Such approaches may be supervised or unsupervised.

#### 4.1.1.5 The Value of HMMs as Language Models

HMMs have many advantages. They can be used and trained efficiently and they have proved themselves in many applications. However, as we have discussed, they have only the equivalent expressive power of regular grammars. Natural Languages are generally considered to be at least context-free. As such, their application as probabilistic models of language is limited and we really need to consider the extension of these approaches for use with context-free grammars, which is done later. However, we will now look at how effective HMMs have been for part-of-speech tagging.



#### 4.1.1.6 Using Stochastic Models for Part-of-Speech Tagging

Up to this point, the discussion has remained rather theoretical as to how these methods can be used for part-of-speech tagging. However, Charniak [33] describes a simple approach. Suppose the set of states is identified with the set of part-of-speech tags. The outputs remain as the words, as the problem is still to be framed as maximising the likelihood of a particular string. It is then possible to find the most likely sequence of states, which in turn is related to the most likely sequence of tags.

The relationship between states and tags depends on the type of model being used. For example, they could just be labelled with the part-of-speech for that word (a uni-gram model), or they could be labelled with the current and the previous part-of-speech (a bi-gram model).

Church [39] provided one of the earliest stochastic part-of-speech tagging systems, training on the tagged Brown corpus and using a tri-gram model (based upon the context of the succeeding two parts of speech) he achieved results of 95–99%. However, this is less interesting to us because of the supervised nature of the training algorithm (a corpus annotated with tags is used in training). Similarly DeRose [53] achieved 96% accuracy on the same corpus using what appears to be essentially an HMM approach.

These approaches provide good results and have again been developed to work in an unsupervised manner. Again however, a lot of information is already supplied. The task is disambiguation only, as a lexicon is available in all the systems. However, the indication is strong that stochastic models can be exceedingly effective on their own as simple language models and simple syntax assigning models. The next step would be to apply such models to assigning more complex syntactic tags, such as the CG categories that are used in CLL.

#### 4.1.2 Unsupervised Part-of-Speech Tagging

There are two main approaches to unsupervised part-of-speech tagging. Firstly, from a symbolic perspective, there is work modifying Brill's transformation-based error-driven learning model to an unsupervised setting. Secondly, it is possible to use the Baum-Welch algorithm described above in an unsupervised way.

The unsupervised version of Brill's model has almost exactly the same structure as the supervised model shown in Figure 4.1. However, instead of the *truth* module, which was a manually annotated corpus, the learner now has only a lexicon. The *initial state* is now set by taking the unannotated text and attaching a list of all possible tags a word may take in the lexicon to each word, which leaves us with the initially annotated text.

Again, transformation templates are used to build transformations that improve the annotation (i.e. remove ambiguity). There are now only four templates for triggering conditions, which, given a word-tag pair, look at the word or tag which precedes it, or the word or tag which follows it. The reduction in transformation templates is presumably to reduce the amount of processing that the system needs to perform. The rewrite rule part of the transformation is now somewhat different. Instead of changing the tag of a word, it removes the ambiguity, by replacing the list of possible parts-of-speech, with which the word is currently tagged, with a single part-of-speech. An example, which was learned [27] of a

transformation in the unsupervised setting is:

Change the tag from NN\_VB\_VBP to VBP if the previous tag is NNS

Here the tag of a word is reduced from a list of three to one particular tag (see Appendix A for a list of the Penn Treebank part-of-speech tags, which are used in this example). Becker [12] has extended the template set and noted that this causes efficiency issues. However, this extension was shown to be effective in achieving higher accuracies with smaller numbers of transformations on a small German corpus.

Perhaps the most important difference between the unsupervised and the supervised versions of the model is found in the building of the list of transformations. In the supervised version it was possible to determine the best transformation as the one that changed the corpus to be most like the *truth*. In this case, the learning is unsupervised, hence there is no definition of the truth. In effect, Brill [27] resorts to a statistical method of scoring possible transformations, using frequency information. Given the transformation:

Change the tag  $\mathcal{X}$  to  $Y$  in triggering environment  $C$

the quality of the transformation is calculated on the basis of the current annotation of the corpus. The underlying idea is to use the unambiguous tagging of the current corpus to determine likely sequences of tags. For each tag  $Z \in \mathcal{X}$ ,  $Z \neq Y$ , the system calculates

$$\frac{freq(Y)}{freq(Z)} \times incontext(Z, C)$$

where  $freq(t)$  is the number of times  $t$  appears unambiguously in the corpus and  $incontext(t, c)$  is the number of times tag  $t$  appears unambiguously in context  $c$  in the corpus. The quotient multiplier provides normalisation, thus removing the effect of the overall frequency of the tag. If

$$R = argmax_Z \left( \frac{freq(Y)}{freq(Z)} * incontext(Z, C) \right)$$

then  $R$  is the most common tag in the context  $C$  of the other possible tags for the word. Hence, the quality of the transformation will be calculated against this possible alternative transformation. The score given to the transformation is:

$$incontext(Y, C) - \frac{freq(Y)}{freq(R)} \times incontext(R, C)$$

Again the quotient is used to normalise the effects of the overall frequency of the categories. The effect is then to take the difference between the occurrence of  $Y$  in the context  $C$  and the best alternative,  $R$ , to get a measure of quality. At each iteration the learner selects the transformation that maximises this function and the learning process continues until no positive scoring transformations exist.

Brill [27] trained the tagger with a 120,000 word training set from the Penn Treebank using the approach described above. A test set of 200,000 words from the Penn Treebank was tagged by the system and the annotated version of the treebank was used as a gold standard against which the system's tagging was compared. Tagging accuracy on the training set is



95.0% and 95.1% on the test set (which suggests the system does not over-fit to the data). 1,151 transformations were learned to disambiguate the initial state.

Brill's approach is particularly interesting, as it is unsupervised and uses both symbolic and statistical techniques to learn rules. It is also interesting that the symbolic methods appear to curb over-fitting to the data, which can be a problem with stochastic methods. It is worth noting that the perspicuity of the results of the training element of the system are exactly what was desired when CLL was designed. Brill's system also uses context to determine the correct tags, which is similar to CLL, especially as the more frequently seen contexts drive the learning process.

There are, however, significant differences to CLL. Again the system is learning simple parts-of-speech rather than the more complex syntactic categories of CLL. The system also uses a complete lexicon, reducing the task to one of disambiguation, rather than actually learning the word-to-part-of-speech mapping. In fact, the algorithm relies heavily on this lexicon. If, for example, all words were assumed to be able to have all parts-of-speech, so no lexicon was needed, then the technique for determining the quality of a transformation could not easily be applied (all would have the same quality). It is the lexicon that pushes the learner in the correct direction. It might be possible to use only a partial lexicon, so long as it contained some examples of words that could only take one part-of-speech, the rest of the words could then take all possible parts-of-speech. Alternatively, the system would have to be modified to allow transformations to reduce the ambiguity of a word rather than eliminate it, then a partial lexicon containing words that could take a subset of all the parts-of-speech may be enough to bias the learner appropriately. However, the most likely result of doing this (especially given the efficiency considerations raised by Becker [12] for a much simpler extension), would be to create an inefficient learning algorithm. However, in future, it may be interesting to investigate this possibility.

A further limitation of the system is that it is not incremental. The corpus is considered and used as a whole. This is clearly not the manner in which a human learner would receive it (although it could be argued that the learner may retain it in and use it as a whole). CLL does deal with examples incrementally, although, as has been mentioned, the fact that seen examples are stored, while not affecting incrementality, is also probably psychologically implausible. In general though, this system is solving a simpler problem, with rather more information than CLL. It is also probably the case that the extra information that has been used (including using the corpus non-incrementally) is not psychologically plausible.

The Baum-Welch algorithm can be used, unsupervised, to perform part-of-speech tagging. However, to achieve reasonable results, some extra background knowledge needs to be provided. Kupiec [78] has used an unsupervised version of the Baum-Welch algorithm. As well as supplying a lexicon, he places the words in equivalence classes to deal with the sparse-data problem. Words in the same equivalence class must take one of a specific set of parts-of-speech. This improves the accuracy of this algorithm (i.e. the unsupervised version) to around 96%, which is similar to the supervised HMM models.

With respect to building the underlying model of a part-of-speech tagging system, it is entirely possible to use Stolcke and Omohundro's model [120] in an unsupervised manner, i.e. part-of-speech tagged data cannot be used in building the initial model, or in determining



how states should be merged.

These two models show that unsupervised approaches are possible, if rare. They also show that it is possible to achieve results that are nearly as good as those of supervised learning on the part-of-speech tagging task. However, it must be noted that both these models required a lexicon as background knowledge. Kupiec’s approach [78] also needed further background knowledge – the equivalence classes. Brill’s approach [27], rather than extra background knowledge, used a further statistical learning element. Hence, there is a trade-off. Reducing supervision will lead to increasing the background knowledge or the complexity of the learning methods (or perhaps both).

### 4.1.3 Supertagging

Abney [1] suggests that the relative success of part-of-speech tagging leads to the question of which part of NL-understanding can be solved next. Some have tried to extend part-of-speech tagging techniques towards a kind of parsing [72]. The underlying idea is that, if the tags can be considered to be complex syntactic categories (“supertags” [72]), then tagging will actually specify a lot more syntactic information. While this tagging is not complete parsing, it is a significant increase in complexity over simple part-of-speech tagging and can be described as “almost parsing” [72, 10].

Clearly, it is wise to use some kind of lexicalised formalism to get a large amount of syntactic information stored in the lexical tag that is learned. Hence, formalisms like CG, HPSG and LTAG are particularly useful in this context.

This idea is really quite close to the approach taken with CLL, where words are given a complex CG tag. The main difference in respect to the results of such a system should be noted at the start. CLL, as well as tagging sentences with these complex syntactic tags, also provides a complete parse unlike these approaches, which can at best be considered to give partial parses.

Joshi and Srinivas [72, 10] present a number of approaches to disambiguating supertags. In this work they use Lexicalised Tree Adjoining Grammar (LTAG, see Chapter 2), where, like CG, the grammar consists of a lexicon where words are assigned complex syntactic categories (elementary trees in this case) and there are simple rules (substitution and adjunction) for combining these trees to get a parse.

Their approach to supertagging involves collecting a lexicon, which contains the set of supertags that can appear with a word and the probability of their appearance. The lexicon is collected from the XTAG parsed version of the Wall Street Journal (which contains sentences from the treebank of at most 15 words). Hence, once again, the approach is supervised and is simply tag disambiguation – a much simpler problem than that which CLL is designed to solve.

Various models are used to perform the disambiguation. Initially, as some kind of baseline, a uni-gram tagging model is used. This simply assigns the most likely tag to the word irrespective of the context. The probabilities are calculated for standard part-of-speech sequences produced by Church’s tagger [39] rather than for words, as the data is too sparse. By using part-of-speech sequences, much of the ambiguity is removed, thus simplifying the problem.



When the uni-gram model is applied to an unseen set of 100 sentences from the WSJ (presumably of at most 15 words) and only the most probable sequence of supertags is used, then only 15% of the sentences can be parsed (this is not a measure of either lexical label accuracy or structural accuracy, but simply a measure of whether a parse is possible). If the most probable three supertags are assigned to each word, then 52% of the sentences can be parsed.

A tri-gram model is then used with the data (a HMM model where probabilities between states are calculated based on the trigram statistics extracted from the XTAG parsed corpus and therefore the approach is supervised). This model achieved 68% success when applied to the same test corpus. Here success is defined as a word being assigned its correct supertag (presumably when compared with the XTAG assigned supertag).

Finally, a model based on maintaining syntactic dependencies between tags was developed, i.e. the model ensured that the dependencies between tags were maintained and calculated the most probable path for maintaining these dependencies based upon statistics drawn from the XTAG parsed corpus. This model is especially interesting, because it essentially applies the syntactic constraints between the supertags in an attempt to improve tagging accuracy, which is similar to the parsing approach of CLL. In this case, 77.26% success is achieved, where again success would appear to be assigning the same supertag as found in the XTAG parsed data.

It seems worthwhile to note what results are not present in this work. Firstly the uni-gram model does not provide a baseline, as a different measure is used to evaluate its accuracy – that of the possibility of getting a parse from the sentence, which contains a rather debatable link to the actual correctness of the tagging. Secondly, the experiments with the tri-gram model and the dependency model do not use this measure, which would be useful. In part-of-speech tagging the percentages of correctly tagged sentences is usually significantly lower than the percentages of correctly tagged words (see e.g. Cussens [48]). Is this the case with these supertagging approaches?

In general, the supertagging venture has much in common with CLL in that it is aiming to label words with complex syntactic categories. However, the setting of the problem is significantly different. The model is supervised both in determining probabilities and also in using a lexicon extracted from a tagged corpus. This is clearly implausible with respect to the principles upon which CLL was built. The use of the lexicon also means that the problem is actually one of disambiguation rather than learning the lexicon. This indicates a much simpler problem, which is exacerbated by the use of part-of-speech sequences in both training and tagging, which removes a significant amount of the ambiguity.

However, the approach shows some useful results. Firstly, simply assigning the most common tag to words seems to provide a fairly low parse accuracy, although it is debatable as to what the tag accuracy is. Secondly, this work shows that a simple stochastic approach that works well for part-of-speech tagging does not perform very well for supertagging and a model that has a much larger symbolic content (the dependency model) performs significantly better. All this said, the system does still only produces a maximum of 77.26% accuracy, which is 20% less than the better part-of-speech tagging systems. This gives an indication of the complexity of the problem which CLL is being used to solve, which is significantly



more complex still than that of supertagging.

Later results [10] are greatly improved using the tri-gram model, but there have been no further experiments using the dependency model due both to lack of resources for training and because the aim of the supertagging approach was to use local constraints rather than the dependencies captured by the model. This is unfortunate, as this model is perhaps the closest to that of CLL. However, the most recent results with the tri-gram model have shown a significant improvement, which is apparently due to the use of a larger training set, smoothing techniques and lexical information. The experiments are however still carried out upon part-of-speech sequences, significantly reducing the ambiguity in the problem and a complete lexicon is still used. However, the results remain impressive with a best accuracy of 92.2%.

## 4.2 Syntax Learning

As with part-of-speech tagging there have been supervised and unsupervised approaches to learning syntax. Again, both the supervised (Section 4.2.1) and unsupervised (Section 4.2.2) are discussed, as both offer valuable insights to the process.

By its very nature, the process of learning syntax must have some symbolic element to it. After all, the notion of syntax, i.e. the structure of language, is symbolic and any stochastic language model that is used will, in general, be applied on top of a symbolic model. In each section, the purely symbolic methods are discussed before those that use stochastic models as well.

### 4.2.1 Supervised Syntax Learning

There are a large range of supervised techniques for learning grammar. In this review, the discussion is restricted to those techniques that are most relevant: learning lexicalised grammars and learning parsers. These two areas are probably the closest to those investigated in this thesis, as the grammars the model of the previous chapters is designed to learn are lexicalised (CGs) and the parsing methodology is used in the learning process.

#### 4.2.1.1 Lexicalised-Grammar Learning

The majority of systems reviewed here learn CGs, as these are the most relevant to the work presented in this thesis. However, I also present some work in learning Lexicalised Tree-Adjoining Grammars (LTAGs) and Head-Driven Phrase-Structure Grammars (HPSGs).

MURRAY is the name of a system, developed by Russell [104], with the aim of inducing HPSG feature structures, containing syntactic and semantic information, whenever new words are presented. The setting is supposed to be similar to that of an adult learning a new word, so that there is both a large amount of syntactic and real world knowledge already available.

Essentially, a sentence containing an unknown word is parsed providing a set of hypothesised feature structures for the unknown word. The feature structures are hypothesised using the scanner within MURRAY's parser that detects the constraints on the feature structure



for the next word. In other words, the context is used to determine the category for the new word, which is similar to the way unseen words are dealt with in CLL. Hypotheses that allow a parse are retained and gradually refined by comparing with other feature structures assigned to the word. The “correct” feature structure is built up gradually in this way.

The use of the parser to apply constraints is similar to the way the parser is used in CLL, however, the setting of the problem is very different, as a large amount of world and syntactic knowledge is available to the system and it is used to drive the process for determining the new categories. In the setting defined for CLL, this amount of knowledge is deemed to be far too great to be practically useful or psychologically plausible in the context presented here (although it should be noted that this is not the case for the adult word-learning approach for which MURRAY is designed).

Xia [145] describes a highly supervised approach of translating the Penn Treebank annotation into an LTAG annotation using the structure present in the treebank and heuristics for determining the head/complement/adjunct structure from the treebank labelling. An LTAG grammar is then extracted from these translated trees. Hockenmaier *et al* [67] have pursued this approach, but applied it to extracting Combinatory Categorical Grammars [118, 144, 119]. These approaches have similarities to the method used for translating the Penn Treebank into a CG formalism, which I use to build corpora for the evaluation of CLL and so they are reviewed in some detail in Chapter 10. However, with respect to their learning approach, while it is interesting that they learn lexicalised grammar, the extreme nature of the supervision indicates that they have little relation to the CLL approach.

An early CG induction method is presented by Kanazawa [73], who developed an algorithm based upon Buszkowski’s discovery procedure for CGs [31]. The algorithm performs identification in the limit receiving positive examples only. The thesis describes two approaches to learning. The first is where the positive examples are simple text strings. This is less successful than the second (as it appears to require exponential time), where the positive examples are text strings annotated with their functor-argument structures (F-structures). F-structures are a form of structural information (i.e. supervision) annotating the examples with both syntactic and semantic structure (due to the correlation between syntax and semantics in CG). The algorithm uses variables for categories and then uses unification to build up the precise categories. It is in fact very similar to the approach used by Xia [145] and Hockenmaier *et al* [67] in using function structure to translate trees and thus build categories, although the F-structures have less information, as they are unordered (whereas the treebank translation systems have ordered trees). However, unlike these systems, Kanazawa’s system [73] has no results reported for real natural language data.

While a number of these grammar learning approaches are interesting, it would appear that none fit the model defined in Chapter 3. However, they show the value of lexicalised formalisms, especially CG, in the natural-language learning context.

#### 4.2.1.2 Parser Learning

A number of people have investigated machine learning of parsers rather than grammars. This has the advantage of allowing the learning process to modify both parser and the grammar at the same time, but perhaps the disadvantage of not allowing a clear separation



between the two.

One approach has been to learn how to make parse decisions effectively. Zelle [146] and Zelle and Mooney [148] have developed an approach using Inductive Logic Programming (ILP) [93] to learn shift-reduce parsers from a set of positive examples using some background knowledge.

The system, named CHILL, takes examples of sentences paired with their desired parse (hence the approach is supervised) and generate a number of *parser operators*, which define what the parser should be doing at each stage of the parse. The initial learning phase produces an over-general parser. This parser is used to parse the training examples, from which process, a set of positive and negative examples of the use of the parser operators is generated. These examples are given to an ILP system, which learns rules for the contexts in which the parser operators can be used. The system has also been used to learn semantic parsers, i.e. parsers that build logical forms rather than syntactic analyses, [147, 123].

This method is clearly supervised, as full structural annotations are provided for all the training examples, which is not the setting of CLL. It is, however, interesting to see ILP used in this context. The approach also learns deterministic parsers, which is somewhat implausible (although perhaps computationally useful). CHILL appears only to have been used with fairly small corpora (e.g. ATIS [88]), which is perhaps not surprising given that ILP approaches can struggle with large training sets. It would be interesting to know how effectively the approach would scale up.

A similar approach that uses decision-structure learning instead of ILP for learning parse decisions, is presented by Hermjakob and Mooney [65]. Here again, the Marcus parser [86] appears to be the goal. Again, only positive examples are used, but these have the set of parse actions attached along with a set of features. The examples are a simplified version of the Penn Treebank (concentrating on examples made up of the more frequently occurring words). The system achieves results of around 92% precision and recall.

The weaknesses of the system with respect to the research presented here are essentially the same as those for CHILL, in that there is heavy supervision and the dubious goal of a deterministic parser. The system does perform well however, using a strongly symbolic method.

An alternative ILP approach has been investigated by Cussens and Pulman [50], where the predictive nature of the chart parser is used alongside an ILP algorithm. The chart parser is used on sentences that cannot be parsed using the current grammar. This produces a list of all possible edges. The set of *needed* edges, i.e. those that could be used to complete the parse, is generated. These are used as positive examples from which an ILP algorithm is used to build possible grammar rules. The rules are evaluated to build some notion of the “best” grammar.

While it might be possible to start with a very basic initial grammar, it is clear that this is intended to be a grammar-extension algorithm rather than a grammar-learning algorithm. The approach has only been tested on small corpora (including the LLL corpus [74] which was used in early experiments with CLL) and so has yet to be convincingly evaluated. However, it is interesting to note the use of a chart parser’s predictive strength, which is similar to the use of the chart parser in CLL.



The final approach to discuss, with respect to learning parsers, is an extension of Brill's Error-Driven Transformation-Based Learning method [24, 22, 23, 111], which has already been discussed with respect to part-of-speech tagging [25, 26]. This time the initial state is built by a parser that simply assigns a right-branching structure (with no labelling) to the examples. The learner then builds a set of transformation rules (as with the part-of-speech tagger), to transform these parses. These rules are restricted to instantiations of twelve schemas for adding and deleting brackets. Learning occurs in the same way as it did with the supervised part-of-speech tagging system. The rules are applied in all cases and then scored and selected with respect to the improvement they cause to the corpus when compared with the correct parse (hence the system is supervised). The system achieves 91% accuracy on the rather small ATIS corpus. A more efficient version of the parser is reported by Satta and Brill [111].

The system is clearly reasonably effective, but results have only been presented on fairly small corpora. The system is also heavily supervised and non-incremental and as such is a rather different setting to that of CLL. It also only learns bracketing and not any kind of labelling, which is one of the aims of CLL. Finally, the system is given the right-branching heuristic. It will be seen in Chapter 11 that this is a strong heuristic for English (an essentially right-branching language). One might expect only a small number of transformations would be needed for the small sentences of the ATIS corpus. CLL is not given this heuristic (although possibly it is given a bias towards it), but appears to learn it.

In general then, the parser learning approaches encourage the use of syntactic analysis alongside syntactic learning, an approach followed by CLL and they indicate that the predictive power of a parser in learning new rules/categories is very useful. However, thus far they have remained either small scale approaches or relatively heavily supervised approaches. With CLL, larger-scale experiments have been performed to build large lexicons without heavy supervision.

Recently, a large number of approaches combining stochastic and symbolic methods have been used. The majority of these approaches take an already defined grammar, add probabilities to it and then train them in some way. However, some, like CLL, build the grammar and the probability model at the same time. In the following survey, parsing and grammatical models are combined, as they are in the literature.

Following on from the success of HMMs in part-of-speech tagging, Probabilistic Context-Free Grammars (P-CFGs) were investigated to determine if their increased expressive power could be harnessed to make a good probabilistic model of natural language.

A P-CFG, according to Charniak [33], is a quadruple  $\langle W, N, N^1, R \rangle$ , where  $W$  is the set of terminal symbols  $\{w^1, \dots, w^\omega\}$ ,  $N$  is the set of nonterminal symbols  $\{N^1, \dots, N^\nu\}$ ,  $N^1$  is the start symbol and  $R$  is the set of rules of the form  $N^i \rightarrow \zeta^j$  where  $\zeta^j$  is a sequence of symbols in either  $W$  or  $N$ . Each rule has a probability assigned to it and the probabilities for all rules with  $N^i$  on the left-hand side must sum to one.

The probability of a sentence is the sum of the probabilities of the parses for that sentence. The probability of a parse is the product of the probabilities of the rules used in the parse (note this is based on certain flawed independence assumptions).

The probability of a sentence can be found efficiently by calculating either the *inside* or

the *outside* probabilities. The inside probability is defined as:

$$\beta_j(k, l) = P(w_{k,l} | N_{k,l}^j)$$

which is the probability of the part of a string  $w_{k,l}$  given it is covered by the nonterminal  $N^j$ . Similarly, the outside probability can be defined as:

$$\alpha_j(k, l) = P(w_{1,k-1}, N_{k,l}^j, w_{l+1,n})$$

As with HMMs, an efficient recursive algorithm exists for calculating the probability  $\beta_1(1, n)$  (see Charniak [33] for more details).

Again, as with the HMMs, the Viterbi algorithm [33], can be used to find the most probable parse for a P-CFG. The approach is to maintain the most probable way of arriving at each node (rather than each state as with HMMs) within the parse tree. In this way improbable parses are removed and so efficiency is maintained.

The Baum-Welch algorithm can be modified for training the P-CFGs [33]. The algorithm is essentially the same as that for HMMs, but the probability re-estimation function is now based on counts of the uses of particular rules.

$$P(N^i \rightarrow \zeta^j) = \frac{C(N^i \rightarrow \zeta^j)}{\sum_k C(N^i \rightarrow \zeta^k)}$$

Again this training has the problems of over-fitting and of only determining a model that is a local maximum in the search space.

Stolcke and Omohundro [120] give an example of their approach to building initial models using P-CFGs and so it is possible in this case to build the P-CFG from scratch, although practically how effective this would be for a large scale natural-language grammar is questionable.

P-CFGs have not been very satisfactory as probabilistic models of syntax [44, 84]. The common criticism is that P-CFGs are not statistically sensitive enough. By only attaching probabilities to rules they do not take history or context into account and so there are flaws in the independence assumptions. There have been a number of approaches to adding statistics in more useful ways.

Probably the most extreme has been that of Bod [18], who, in building a grammar for parsing, has developed Stochastic Tree-Substitution Grammar (STSG). Here, the grammar is a set of all subtrees which can be extracted from a syntactically labelled corpus.

These subtrees can then be joined to build parses by *substitution* where the leaf of one tree can be replaced by a second subtree if the leaf of the first tree is the same non-terminal as the root of the second tree (Figure 4.6 contains an example). The subtrees are assigned probabilities from their relative frequencies in the corpus from which they are extracted. The probability of a parse is the sum of the probabilities of all ways of building the parse. The probability of a particular way of building a parse is the product of the probabilities of the subtrees used.

This type of approach has been used very effectively for parsing with the Data-Oriented Parsing (DOP) model [18, 19]. However, it would intuitively seem that there are too many



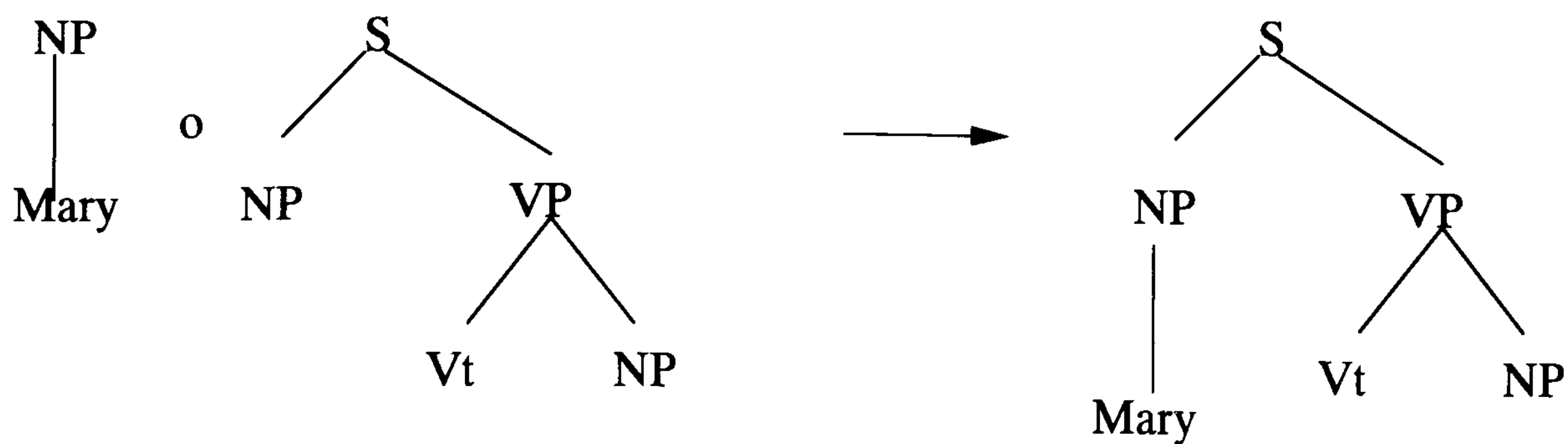


Figure 4.6: An example of substitution in STSG

probabilities being calculated, which in some sense is confirmed by the fact that no efficient parser has really been developed for this model, rather a Monte Carlo method has been applied to build parses.

Most recently Charniak [35], has taken a more restrictive approach to adding further context, by including information on the immediate head to determine probabilities more accurately. Similarly both Collins [44] and Magerman [84] follow a more restricted approach to adding statistics by selecting the features to which probabilities will be added. They add, for example, primary and secondary lexical heads [84], and dependencies and direction with respect to the head [44]. These algorithms achieve much better efficiency than the DOP approach and approximately equivalent results of about 90% precision and recall on the Penn Treebank.

All of these approaches are essentially lexicalised, as such approaches are currently proving to be the best for parsing [35]. However, the grammar models are in general to be extracted from and trained using annotated corpora, making the learning process heavily supervised. However, they do show that a lexicalised approach appears to be the best for syntactic analysis and that carefully used probability models would appear to be very useful for good parsing. Similarly, CLL uses both lexicalised and stochastic grammars.

Osborne [95] and Osborne and Briscoe [96] present work that bears some similarities to CLL. Their system learns CGs from sequences of part-of-speech tags (hence the system is somewhat supervised) using MDL. The system builds binary-branching trees using Bayes' rule to estimate the most likely trees, where a tri-gram model is used to estimate the probability of the data. This tri-gram model is built using annotated data, i.e. the process is supervised, and also provides extra clues as to the structure, given that probabilities of different sequences are higher if they are constituents. Hence, as the authors admit, the approximation to the probability of the data should perhaps be dropped<sup>1</sup>. The trees are built bottom-up. A pre-defined table is used to determine the root label of each subtree. Such a table, actually again provides a degree of supervision, which is not far removed from providing structurally annotated data. These trees are then used to derive the categories in a very similar way to Xia [145] and Hockenmaier *et al* [67] using the functor-argument structure to determine the complex syntactic categories.

The approach is supervised (although perhaps less obviously than some of the approaches above) and learns rather unusual grammars, as they are CGs with a large number of part-of-speech tags as the atomic categories. They do not appear to take any account of movement,

<sup>1</sup>Thanks to Miles Osborne for his explanatory comments on this process.

i.e. the grammar would be unlikely to produce very elegant analyses for sentences containing movement. However, it performs well with respect to bracketing, returning a crossing-bracket rate of around 3 when trained on the British National Corpus and tested on the Spoken English Corpus<sup>2</sup>. However, recall and precision values are both around 50% (presumably for the bracketed constituents), which may seem surprisingly low given the crossing bracket rate. However, as is discussed in more detail in Chapter 10, binary-branching trees will give poor results with precision at least, as such trees tend to hypothesise many more brackets than are contained in annotated corpora.

## 4.2.2 Unsupervised Syntax Learning

In this section on unsupervised syntax learning, a broad range of techniques is discussed, as these approaches are the closest to those that are investigated in this thesis.

### 4.2.2.1 Learning Rule-Based Formalisms

There have been a variety of purely symbolic approaches to learning syntax. Some of these have concentrated on working within the Identification-in-the-Limit framework of Gold [60] (discussed in Chapter 3). Due to his negative results with respect to unsupervised learning they have concentrated on restricting the problem to make it possible.

For example, it is possible to restrict the search space of languages which the learner can investigate, i.e. to learn subclasses of languages. This can be achieved by restricting the types of rules that can be learned. A prime example of this technique is the method of identifying  $k$ -reversible languages (a restricted sub-class of regular languages) in the limit using Angluin's automata induction algorithm [5]. This algorithm has been used by Berwick and Pilato [15] to learn finite-state automata for parts of English (noun-phrase modifier constructions and auxiliary-verb constructions) and also by Watkinson [133] to learn automata for musical structure. Given a finite set of examples, the algorithm can be shown to terminate in polynomial time in the length of the example set.

The algorithm for building the finite-state automata is entirely unsupervised – the only input is the example sentences – which makes it particularly interesting from both a computational and a psychological perspective, as has been discussed. It shows that it is possible to learn parts of syntax with very little background knowledge. However,  $k$ -reversible languages are regular, and so do not have the expressive power to capture a large amount of the subtlety of natural language. Hence, the algorithm is perhaps effective because the class of languages has been restricted so much. Berwick and Pilato [15] argue that this is not a serious problem, as there may be a whole collection of different learning methods for learning different parts of natural language. However, this does not really address the problem as to what methods are used to learn the context-free (or even context-sensitive) parts of grammar. More generally, this approach does not really address many of the issues of the environment in which learning occurs, what may or may not be background knowledge and what exactly needs to be learned.

---

<sup>2</sup>Currently the web page <http://info.ox.ac.uk/bnc/> contains information on the British National Corpus and links to information on various other corpora



Sakakibara [105, 106, 107, 108] has extended this approach to allow the set of *reversible* context-free grammars to be learned. This might seem attractive, but unfortunately, to build the grammars, the examples from which the learner generalises must have structural information. This is the other approach to getting around the restrictions of the Identification-in-the-Limit framework. The search space of the grammars can be reduced by providing extra information.

In Sakakibara's approach [105, 107] an unlabelled parse tree annotates each example (i.e. the examples are bracketed and the learner learns the labels for the tree). Such examples are commonly not available and are expensive to produce.

In fact the constraints that these systems have to place upon the space of grammars suggests more about the framework within which they are working. As suggested in Chapter 3, the framework of Identification in the Limit is almost certainly too restrictive and is not very accurate with respect to the learning environment that I am aiming to model. This has led to the abandoning of this as a framework in which to investigate language learning on the whole. It is more common to use more flexible frameworks and this has been our approach throughout the development of CLL.

An appealing alternative is the use of compression to generalise. Wolff [143, 141, 142] has pioneered this, especially with respect to language learning. The systems he uses are designed to learn rule-based grammars in a psychologically plausible way. The aim is to identify patterns from unannotated text and use the best of those patterns to generalise, forming first constituents and then grammar rules using these constituents. The text is, therefore, compressed into a grammar representation. This approach has similarities with CLL, for example the use of compression based upon syntactic analysis and the suggestion that this is a psychologically plausible method.

Unfortunately, the system would not appear to have been used on anything but toy examples and while it has been shown to be effective, there is little evidence that the system will scale up to be able to deal with full natural language corpora. In fact, it would appear that the process of determining constituents could be computationally very expensive. Hence, the system may be somewhat impractical for large scale experiments. However, the strong argument for compression being a psychologically plausible method for language learning has been influential in the design of CLL.

#### 4.2.2.2 Acquiring Parameters

An alternative to learning PSG-based grammatical formalisms, has been to assume the Chomskyan grammar theory of Principles and Parameters (see Chapter 2). The Principles and Parameters theory has not been selected as the theory of grammar for CLL, which was discussed in Chapter 2. The main reasons for not selecting it being that the grammar was imprecise both as to the correct principles and parameters involved and to the interactions of these principles and parameters and the theory does not deal with the lexical knowledge already available to the learner. As such, these methods are of limited interest. However, there are two reasons for providing brief descriptions of some of the better known methods. Firstly, they may provide some interesting insights into the type of techniques that can be used in solving NL learning problems. Secondly, they will also illustrate the problems caused



by using the Principles and Parameters formalism.

Various approaches have been investigated, but in this section some of the better known approaches will be considered. Firstly, there is the method of *triggering*, generally based upon the work of Gibson and Wexler [58] and reported by both Bertolo [13] and Atkinson [7].

Assuming a set of parameterised principles of language, then the parameters are initially set randomly. Each example in the environment is parsed with this setting until an example cannot be parsed. At this stage the learner is “triggered”. The current setting of the parameters in the grammar is modified by randomly selecting a parameter and then randomly selecting an alternative setting for that parameter. This new grammar is adopted only if it allows the example which failed with the previous grammar to be parsed. For example, suppose the current setting of the order of the first schema of  $\bar{X}$ -theory was :

$$XP \rightarrow X' \text{ Spec}$$

which means that Specifiers (or somewhat less accurately subjects) in a phrase appear after the head words. Using this schema with sentential phrases it would mean that subject noun phrases appeared after the verb phrase, e.g. “ran John”. Then suppose an example came from the environment: “John ran.” With the current parameter settings, the parser would fail. If the the above parameter of the principle was selected and changed to:

$$XP \rightarrow \text{Spec } X'$$

then this would account for the error and the new setting of the parameters would be accepted.

This approach, while perhaps in some way modelling a child’s theory revision and being fairly simple, is clearly rather undirected and inefficient. More importantly, to perform the parsing a complete (if simple) lexicon must already be available. No discussion is presented on how this is achieved, or what format this should take. This precludes any discussion of the effects of the words on the direction the parsing/learning can take. Given that it is assumed that the child/system has already achieved the learning of the lexicon, it is rather implausible and unwise to then ignore its existence. More seriously still, an accurate definition of the principles, parameters and their interaction has not yet been achieved [55]. What if two parameters need to be changed at the same time to allow a particular construction to be parsed? Similarly, what happens if changing one parameter allows the parsing of the current example, but will cause the parsing of both previous and future examples to fail? In fact, the issues of parameter interaction could well cause this algorithm to fail to terminate, or at least to get stuck at local maxima in the search space. In fact, Frank and Kapur [55] show that the algorithm can get caught in local maxima, or it can be caught cycling through two or more non-optimal parameter settings, depending on the parameter space.

One approach to solving some of the problems of the triggering algorithm is to assume that the parameters are independent [109]. This results in an algorithm that has better convergence. However, as yet, it would appear to be uncertain as to whether the assumption of independence is correct.

A second, purely symbolic, approach to acquiring the correct parameters is *cue-based*



Sequence	Parameter Value
NP V	$XP \rightarrow \text{Spec } X'$
V NP	$XP \rightarrow X' \text{ Spec}$
NP P	$XP \rightarrow \text{Spec } X'$
P NP	$XP \rightarrow X' \text{ Spec}$

Table 4.3: An example cue-based learning look-up table

learning [13]. Essentially, a table of cues is defined such that strings are matched with specific values of a particular parameter. The occurrence of these strings within the environment is then used as a cue for setting the parameter to that value. A simple, if rather incomplete and inaccurate example, would be to have Table 4.3 as the cue table for the first  $\bar{X}$ -schema order. On finding an example containing one of the sequences in the left-hand column, the algorithm would set the parameter to the corresponding value in the right-hand column.

Clearly, cues could be exceedingly complex and hard to disentangle. Atkinson [7] presents an approach to use statistics to handle this. He suggests that a certain threshold frequency of cues must be crossed before a parameter is set. This is to prevent problems where special cases could cause over-general parameter settings. Clark [42] suggests a similar approach with a more complex model of frequency.

Another approach to solving the parameter setting problem has been to use Genetic Algorithms [13, 41] (which can overcome the problem of interacting parameters). The problem with such an approach, other than the lack of a well-defined parameter space for natural language, is that the definition of the fitness function (i.e. the function for defining the quality of a grammar) can be difficult and can have a profound effect on the success of the algorithm.

Clark [42, 43] has also investigated statistical approaches to setting parameters. His suggestion is essentially that parameters are set depending on the frequency of examples of a particular parameter shown to the learner. The intuition is that a learner is shown a set of data where the examples show parameter settings ambiguously, i.e. there may be a number of possible combinations of parameters that license the structure of the data. The learner collects frequencies of the possible parameter settings and when a certain threshold is exceeded, then the parameter is set to that value. This may overcome the problem of interacting parameters, but introduces a new problem of how to determine what the thresholds should be.

All these models are essentially unsupervised, although they require some grammatical (at least lexical) background knowledge to set the parameters. The essential problem of not being able to build a realistic model of the principles and parameters remains. Hence, all these models remain untested with respect to real data.

#### 4.2.2.3 Learning Lexicalised Grammars

Given that CLL uses a lexicalised grammar (CG), it is perhaps the systems which use unsupervised learning algorithms for lexicalised grammars that are the most interesting.

Solomon [117] describes an unsupervised approach to learning CG lexicons. It is a deterministic algorithm for assigning categories to words in a corpus of childrens' books of



increasing linguistic difficulty. As with CLL, this approach uses an initial lexicon to bootstrap the process. The algorithm uses a heuristic to determine which word should be assigned a category next and then uses the functional nature of CG to determine the category of that word (where there is choice, the simplest category is chosen, which has some similarity with the compression-as-learning approach followed in CLL). The implications of this choice are then applied to the rest of the corpus.

This algorithm is interesting on a number of fronts. It is unsupervised and it would appear to be efficient (due to its heuristic and deterministic nature). It also demonstrates the elegance of the CG formalism for this sort of task, as it is easy to build categories up given a good enough context. However, there are weaknesses. The testing has been on very simple corpora and it is doubtful that the approach would scale up very effectively, as the author admits. The determinism of the algorithm is particularly concerning with respect to scaling up, as the algorithm would not permit any mistakes. The algorithm is also not incremental in any sense, as it moves over the whole corpus to find the best word to process next (in a somewhat similar way to Brill's error-driven transformation-based learning approach).

Kanazawa [73] also modifies his supervised algorithm (described above) for learning CGs, to allow it to act as an unsupervised algorithm. However, this approach would appear to be too inefficient to be practical. There are further problems, as the algorithms do not appear to have been tested upon natural-language corpora, hence computational efficiency and linguistic appropriateness remain unknown. This is particularly concerning with respect to the kind of grammars that are learned by the system. To prove that the algorithm allows learning in the limit, it was necessary to restrict the type of grammars that can be learned – in particular the number of categories that are allowed per word. These are called  $k$ -valued grammars and it is unclear whether the algorithm would learn linguistically plausible grammars under these restrictions.

Yet another approach to CG learning is the EMILE system originally developed by Adriaans [2]. EMILE builds CF-PSG grammars using the CG framework as a kind of intermediate representation, used to express the functor-argument structure of words. In effect the algorithm traverses a sample of sentences looking for similar contexts from which constituents can be identified. When all potential constituents have been identified, they are tested using substitution into other examples. An *oracle* is questioned to determine whether or not these substitutions are legitimate. Hence this is query learning, which was discussed in Chapter 3. These constituents are then clustered and folded into each other to give appropriate CG categories, which are then rewritten as rules. Such an approach again demonstrates the power of the CG formalism for representing the kind of information that is useful in learning. However, the use of the oracle is, given the model of Chapter 3, impractical in this context.

However, a more recent version of EMILE [129, 128] does not use the oracle. It is still based upon the substitution of constituents, but it appears that, after identifying all the possibilities, a more powerful rule induction approach is used to build an appropriate grammar. This version of EMILE has been tested on a variety of corpora including OVIS [127] and ATIS [88] (which is part of the Penn Treebank). It produces very good results with respect to crossing brackets measures (see Chapter 10 for a discussion of the crossing-brackets



measure) – less than one on average – although it should be remembered that the sentences are fairly short in these two corpora. However, the unlabelled recall values (i.e. the number of bracketed constituents discovered) that are obtained are interesting. The recall values are significantly below 40% in both cases, which suggests that the system is very conservative in its assignment of structure – especially in comparison to such shallowly annotated corpora. Hence the good crossing-brackets results are essentially due to this conservative approach to assigning structure. Clark [40] notes that the crossing-bracket measure can be somewhat flattering to these conservative algorithms. Again it can be noted that the approach is by no means incremental, as it ranges over the full corpus to build up the contexts. There is also no evaluation with respect to the kind of labelling that occurs.

#### 4.2.2.4 Learning Parsers

Perhaps the earliest approach to learning parsers was that of Berwick [14]. His aim is to learn the Marcus parser [86], which is a deterministic parser (it is described briefly in Chapter 6). He also sought a psychologically plausible way to do this and in fact the model is very similar to that proposed here (both in Chapter 7 and Chapter 3). In fact, the approach maintains the further desirable constraint of not using previously seen examples. The learning is achieved by building the IF... THEN constrained parse rules of the Marcus parser as the system comes to examples where new rules are required, which again bears similarities to the approach followed by CLL. It could really be argued that the approach doesn't build a parser, but the grammar used by the parser, however these are rather intertwined.

There are some problems with the system. Firstly, only simple examples are used for learning, which is not psychologically plausible. Secondly, more semantic information than seems plausible is provided to the learner given the discussion in Chapter 7. Thirdly, only a subset of English grammar was attempted (as with CLL movement was not considered as well as other areas), however this may, in part, be due to the fact that the system was being tested on much slower machines. Finally, the Marcus parser, as discussed in Chapter 6, is not a very plausible model for parsing. However, the effectiveness of this early approach had a great impact.

#### 4.2.2.5 Stochastic Methods

Recently quite a number of stochastic approaches have been used for syntax learning. Most of these approaches have been methods for stochastically clustering constituents.

Van Zaanen has recently developed the Alignment-Based Learning (ABL) approach [127, 128]. This has strong similarities to the EMILE system [2, 129], which was discussed above. The same idea of substitution is used to determine possible constituents, however a probabilistic model is used to select those which are likely to be the “best”, hence there is the combination of symbolic and statistical methods. The system performs better than EMILE with respect to the recall and precision of constituents from the OVIS [127] and ATIS [88] corpora, although values are still low (a maximum of about 61% for both) considering the complexity of the corpus. This again suggests that the approach is perhaps somewhat conservative. The crossing bracket rate is around 2.1 for the ATIS corpus. It would be

interesting to see how these systems would perform on the Penn Treebank, when dealing with more complex examples.

The disadvantages of the ABL approach are essentially the same as those of EMILE. It is not psychologically plausible as it does not deal with examples incrementally. However, the algorithm has the great advantage of being unsupervised and so it is an interesting comparison.

Two recent systems have highlighted the possibility of using clustering algorithms for building constituents. Klein and Manning [76] use an entropy based clustering method to build constituents and thus grammar rules. Currently results only indicate that it can learn some useful grammar rules.

Clark [40] uses a mutual-information-based clustering approach with MDL to build P-CFGs. This system has been tested on ATIS and the BNC with reasonable results. Again there is low crossing-brackets rate 0.82, but again the system would appear to be conservative with an unlabelled recall value of 23.7%. It should also be noted that this system learns from part-of-speech strings rather than words, hence making it somewhat supervised.

These systems are interesting because of the weight that they give to the probabilistic methods for building the symbolic grammars. They can perhaps be viewed as attacking the problem the other way around when compared with the majority of the systems above, including CLL, where the probabilistic information is added to a symbolic representation.

Following this discussion, it is clear that there are really very few systems that learn syntax that are unsupervised and use essentially symbolic methods with simple stochastic extensions (most have either complex stochastic models or no stochastic model).

### 4.3 Conclusions

In conclusion it is clear that there is a very large body of research related to learning syntactic information. CLL has been shown to be related to both part-of-speech tagging (especially supertagging), grammar learning and parser learning. The research discussed above has shown that it is sensible to combine symbolic and stochastic techniques and to combine syntax learning and syntax processing.

These systems have not, in general, met the criteria set out at the start of this thesis however. Many are supervised and many of those that are unsupervised use complex stochastic models to make up for the lack of supervision. In the work presented in this thesis, an unsupervised system, using an essentially symbolic process augmented by some simple stochastic methods has been pursued for learning lexicalised grammars from broad coverage corpora. The systems that are closest to this are the supervised approaches of Srinivas and Joshi [72, 10] and Osborne and Briscoe [96], as they learn suitably lexicalised grammar from similar corpora and then the approaches of Clark [40], Van Zaanen [127, 128] and Adriaans and Vervoort [2, 129], which learn very different formalisms, but in an unsupervised way over similar corpora. It will be these systems against which CLL will be compared.



## Chapter 5

# The Learning System

The previous chapters have presented the motivation and a fairly specific model for the type of language learning system that is to be built. In this chapter, we present one possible approach to instantiating the model, the Categorical Lexicon Learner (CLL), which we have used to perform a large variety of experiments in CG lexicon learning and corpus parsing.

Here, the aim is to describe both the learning setting and the learner in detail. The intention is to explain what algorithms have been used and why. Also, the system has a large number of parameters, which will be highlighted and explained. In general, the parameters have been included to allow for empirical investigation. The following description will motivate those investigations. Figure 5.1 provides a diagrammatic representation of the learner.

Section 5.1 contains a description of how CLL can be seen as an implementation of the computational language learning model described in Chapter 3. In Section 5.2 the input

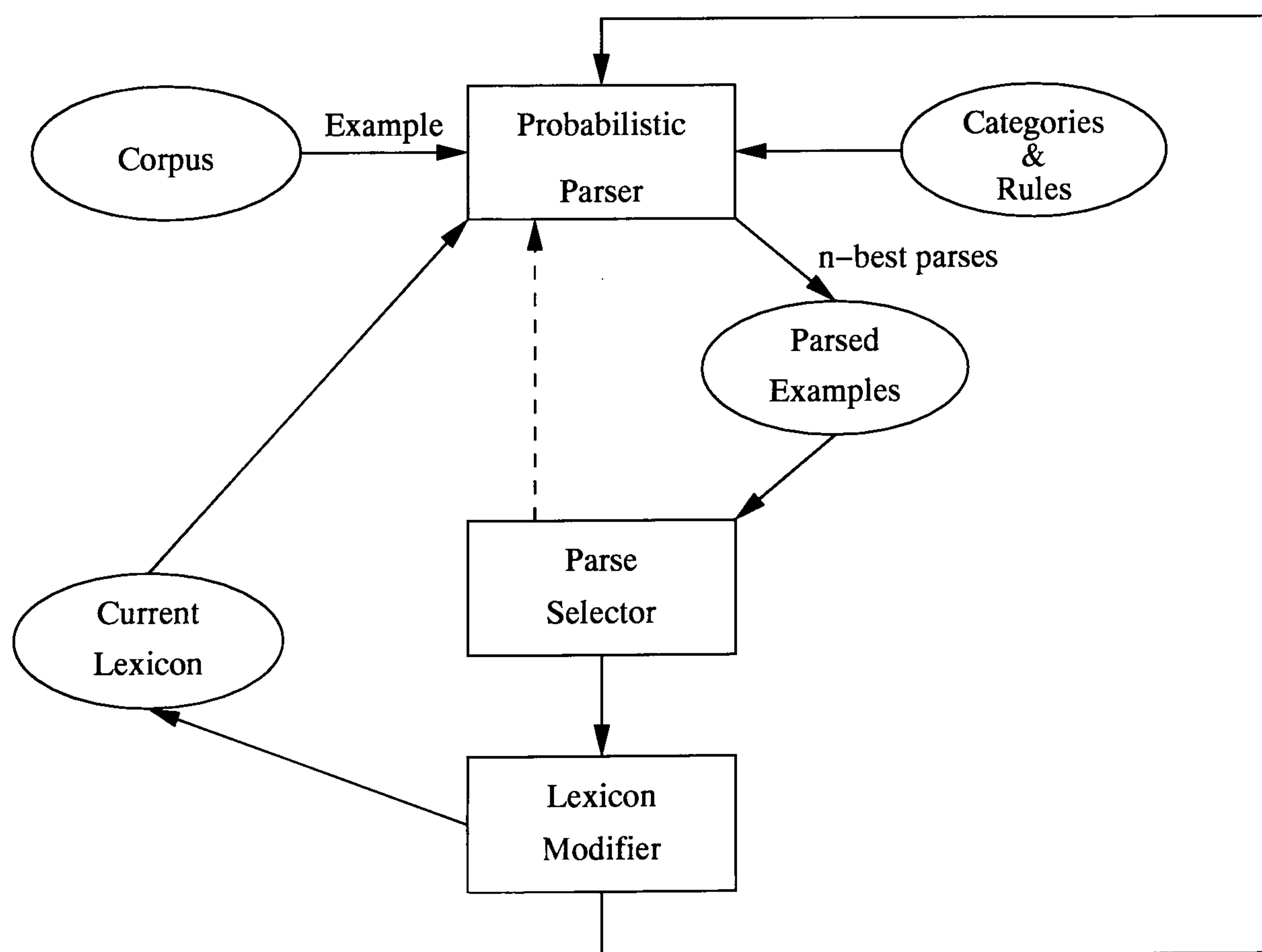


Figure 5.1: A diagrammatic representation of CLL

to the system is described in detail, along with other possibilities. Section 5.3 contains a description of the learning algorithm used in CLL, which is split down into each of the major modules used in the system. In Section 5.4, the output of CLL is described. To help clarify the process, Section 5.5 presents a worked example of how the system processes a particular example given a specific current state.

At this stage, it is probably useful to note that the majority of the system has been implemented in Prolog (in fact, early versions of CLL were implemented entirely in Prolog); however, the parser module is currently implemented in Scheme, which means some communication and translation of data between the Prolog and Scheme parts of the system is needed. The parser is discussed in some detail in Chapter 6 and also in the section on parsing (Section 5.3).

## 5.1 Implementing the Computational Model

Figure 5.1 shows how the computational model in Chapter 3 has been implemented. In this section, each of the elements of the computational model shown in Figure 3.1 will be taken in turn and a brief description of how they relate to the implementation of CLL shown in Figure 5.1 will be given.

The examples in the computational model are provided in CLL by the corpus. Section 5.2 and Chapter 8 describe the types of corpora used with CLL and the format which they take. However, at this stage, it is sufficient to note that the examples in the corpora supplied to CLL are sentences represented as lists of words, where the words are either unannotated or weakly annotated.

The output of the computational model was described as needing to be at least a CG lexicon. For CLL, the output is a probabilistic CG lexicon and a CG annotated version of the examples. These are represented in Figure 5.1 by the current lexicon and the parsed examples. How these are built is described in detail in Section 5.3.

The search engine in the computational model is implemented by two modules in CLL: the probabilistic parser and the parse selection module. The probabilistic parser produces a set of possible lexical assignments for the words of an example and the parse selection module chooses one of these to be used as input for building the lexicon. These modules are both described in detail in Section 5.3. Chapter 6 contains a description of the decisions made when deciding upon the most appropriate parsing algorithm for CLL.

In the computational model, a set of constraints were used to both reduce the size of the search space and to focus the search engine on areas of the search space that are likely to produce good CG lexicons, i.e. the constraints provided the bias. The linguistic constraints of the computational model are provided, in part, by the categories and rules of CG that are given to CLL. Section 5.2 includes a detailed description of these constraints. The system is also provided with some lexical knowledge either as an annotation on the examples in the corpus, or as an initial bootstrapping lexicon (or both). Section 5.2 describes both these knowledge sources. The final linguistic constraints, which are also the history constraints, are provided by the lexicon and parses that are built during the learning process. The way in which these dynamic knowledge sources are used to inform the learning process is described



in Section 5.3.

Two cognitive constraints on the search engine were also suggested. Firstly, processing constraints, i.e. the efficiency constraints, were proposed. These are ensured by the use of efficient algorithms in the construction of CLL, especially the parser. Secondly, the psycholinguistic constraints of approximate incrementality and determinism were suggested. The incrementality constraints are met in CLL in two ways. Firstly, each example is dealt with in turn. Secondly, the examples are processed by the parser in an approximately incremental way. The application of the cognitive constraints is discussed in more detail in Section 5.3 and Chapter 6.

The final type of constraints on the search engine in the computational model were theoretical constraints. Maximum-likelihood and compression heuristics were discussed as possible approaches for implementing these constraints. In CLL the parser is probabilistic and so uses probabilistic constraints to reduce the number of possible lexical entries that could be added to the learned lexicon. Then in the parse selection module, compression is used to select the additions that will actually be made to the lexicon. Section 5.3 describes how these constraints are applied.

Hence, CLL is an accurate implementation of the computational model presented in Chapter 3. In the following sections, each of the parts of the implementation, which have been described briefly here, are described in detail.

## 5.2 The Input

CLL can be considered to have four inputs, which are shown in Figure 5.1 with the oval boxes. Each of these is described in turn below, including the different settings that are used and why.

### 5.2.1 The Corpus

The types of examples contained within the set of corpora that are used for the experiments are described in Chapter 8. Similarly, the system used for building the corpora is also described in Chapter 8. In this section, the actual forms of the input will be described and motivated.

In Chapter 8 two types of example corpus are described. In the first, each example is simply an unannotated list of words. In the second, each example is annotated to indicate which words act as nouns and which as verbs.

The corpus of examples supplied to CLL is a set of Prolog facts, each containing one example as a list of words. Figure 5.2 shows a set of examples.

```
ex(['Sony', answered, the, empty, threat, with, its, real, suit]).
ex([it, '-s', also, costly]).
ex([the, 'bond-s', yield, dropped, to, '7.82_%', the, lowest, since,
'March_31_1987', according, to, 'Technical_Data_Global_Markets_Group']).
```

Figure 5.2: Some sample unannotated examples provided to CLL

```

ex([('N','Sony'), ('V',answered), the, empty, ('N',threat), with, its,
real, ('N',suit)]).
ex([it, ('V','-s'), also, costly]).
ex([the, 'bond-s', ('N',yield), ('V',dropped), to, ('N','7.82_%'),
the, lowest,since, ('N','March_31_1987'), ('V',according), to,
('N','Technical_Data_Global_Markets_Group')]).

```

Figure 5.3: Some sample annotated examples provided to CLL

Similarly, the annotated examples consist of Prolog facts with extra annotation to indicate nouns and verbs. Again, there are various options for this annotation. Words simply need to be annotated with the information that they are nominal or verbal. This is easily achieved by annotating them with an N or a V respectively. Ideally, the annotation should be grouped with the item it is annotating, such that while they are separate entities they are clearly related. It seemed sensible to use the function structure provided in Prolog. While one could use an explicit functor, this would only add to the size of the example files and would provide no information. Instead, I chose to use parentheses grouping a word and its annotation together into a pair. Words that do not have noun or verb annotation, are left without any annotation in the examples corpus. No parentheses are used anywhere else for annotating the examples and it is clear which words are annotated and with what. Figure 5.3 shows the same set of examples as those in Figure 5.2, but with noun and verb annotation.

Figures 5.2 and 5.3 also show the modifications that have been made to the Penn Treebank data. These modifications are discussed in more detail in Chapter 8, however, it can be seen that adjacent nominals are joined by the ‘\_’ character, e.g. ‘7.82\_%’, ‘March\_31\_1987’ and ‘Technical\_Data\_Global\_Markets\_Group’. For convenience apostrophes are replaced with the ‘-’ character, as the apostrophe is a special character in both Prolog and Scheme. The examples show the word ‘-s’, i.e. ‘is’ in this context. Similarly, there is also the possessive ‘bond-s’. All other punctuation has clearly been removed. Finally, it should also be noted that the first letter of words at the start of sentences is de-capitalised where the word is not a proper noun.

While the content and method of building the corpora are discussed in detail in Chapter 8, it is perhaps appropriate to summarise the nature of the corpora used in the experiments here. Two types of corpora are used. Firstly there are machine-generated corpora, where a grammar is used by a computer system to generate examples of the language. Three such corpora were used with CLL. The first two, GC1 and GC2, were generated from context-free phrase structure grammars. The third is the LLL corpus [74], which is a corpus specifically developed for testing language-learning systems. All these corpora are generated from grammars that represent some fragments of English syntax. Table 5.1, in which lengths are measured as words, summarises these corpora.

Secondly, examples extracted from the Penn Treebank [88] were used with CLL. These can be considered to be naturally occurring corpora, as they are text extracted from the Wall Street Journal newspaper. Three versions of this corpora were used, PC1, PC2 and PC4. The first has some noise in and the second and third do not. PC4 is a corpus including longer examples. The corpora are summarised in Table 5.1 where lengths are again in terms



Corpus	Number of Examples	Average Length	Maximum Length
GC1	500	4.14	9
GC2	500	8.72	32
LLL	554	6.30	12
PC1	5000	8.74	15
PC2	5000	9.56	15
PC4	5000	12.54	25

Table 5.1: Summary of the corpora used with CLL

of number of words.

### 5.2.2 The Rules

The Categorical Grammar (CG) formalism is discussed in detail in Chapter 2. A description of the rules, the motivation for using them and examples of their use are included in that chapter. In this section the rules that are used by CLL are stated, including how they are defined to include the probabilities and which are optional. It should be noted that, along with the categories of CG, the rules determine what parses of sequences of words are possible. This means that the rules are integral to the definition of the search space of parses for examples and thus the search space of possible lexicons.

Firstly, the probabilistic forward and backward *functional application* rules (PFA and PBA respectively), which are basic to the CG formalism, are included in the grammar in all of the experiments. These are defined below where  $X$  and  $Y$  are CG categories and  $P_{X/Y}$ ,  $P_{X \setminus Y}$  and  $P_Y$  are the probabilities of the categories they are paired with, i.e. they are a real number in the range  $[0, 1]$ . The probability model for CG is defined in Chapter 2.

$$(X/Y, P_{X/Y}) (Y, P_Y) \Rightarrow (X, P_{X/Y} \times P_Y) \text{ (PFA)}$$

$$(Y, P_Y) (X \setminus Y, P_{X \setminus Y}) \Rightarrow (X, P_Y \times P_{X \setminus Y}) \text{ (PBA)}$$

Secondly, there is the rule for combining noun phrases. This is not included in all the experiments. The rule combines adjacent noun phrases in a sentence to give a compound noun phrase with a probability of the product of the probabilities of the noun phrases that make it up. The rule is defined below, where  $P_1$  and  $P_2$  are probabilities, i.e. real numbers in the range  $[0, 1]$ .

$$(\text{np}, P_1) (\text{np}, P_2) \rightarrow (\text{np}, P_1 \times P_2)$$

It is entirely possible to characterise the CG in a different way and to use different general rules. In future this will be necessary to add rules that allow movement to be handled elegantly. As will be seen, any lexicalised formalism could easily be used within the context of CLL. Ideally, the use of, for example, LTAG and HPSG should also be investigated.

### 5.2.3 The Categories

CLL is supplied with a complete set of CG categories that may be assigned to a word. These must be available both for building the lexicon and for parsing, which in turn means that it needs to be possible to represent them in both Scheme (the language of the parser) and in Prolog (the language of the rest of CLL).

Ideally, from both the practical and the psychological perspective, this set of categories would either be generated dynamically, or perhaps be based upon a set of category schemas, as this would more effectively allow CLL to learn different languages. This area could be investigated further. However, it would seem a reasonable simplifying assumption for the moment to provide the set of language-specific categories that the learner can use, and it is practical, as for different languages it is only necessary to build an appropriate database of categories – a fairly small job. The set of categories obviously defines the search space of the parser when assigning categories to words and thus constrains the search space of possible parses to those containing words with these categories.

While the left associativity of the slash operators in CG is often assumed, it was decided to make the structure of categories explicit with bracketing. This keeps the structure clear for the reader and means that the structure is explicit within both the Prolog and the Scheme code.

The Prolog representation was determined first, as earlier versions of CLL were written entirely in Prolog. In CLL, function structure, is used to build the categories. A prefix representation is currently being used where `fs` and `bs` are used for `/` and `\` respectively. Hence, the representation of the transitive verb category becomes `fs(bs(s, np), np)`.

It may have been noticed that I have assumed that atomic categories will be represented as the atoms `s`, `np` and `n`. This seems the most obvious approach to take, as there is no advantage in representing them as strings.

Several databases of categories are used for the experiments. These databases are, in effect, simply a list of facts, where each category *Cat* has its own fact of the form:

`cat(Cat).`

The construction of the databases is described in Chapter 9 and they are listed in full in Appendix C, which therefore contains many more examples of CG categories represented in this form. Included in these databases is a fact `cat_num(N)`, where *N* is the number of categories available to CLL. It is convenient to have this value (which could be calculated) to determine the probabilities of different category assignments to words.

This is a complete description of the categories and the category databases that are used within the Prolog component of CLL. However, the categories need to be translated into Scheme for the parsing stage of CLL. The parsing is discussed in more detail both in Chapter 6 and below. At this stage it should suffice to state the category representation used. Scheme, like its forerunner Lisp, essentially uses lists for data structures, so a list representation of the categories is logical. In effect, the Prolog structure is followed, but using a Scheme list instead of functional tuples. Hence a category is a triple:

*(ResultCategory Directionality ArgumentCategory)*



Again this follows Steedman's convention [118]. Some examples of categories written in this way, along with the type of role they fulfil are shown in Table 5.2. As before, the atomic categories of CG, *s*, *np* and *n*, are represented in Scheme as the atoms *s*, *np* and *n* respectively.

<i>s</i>	sentence
<i>np</i>	noun phrase
<i>n</i>	noun
( <i>s</i> \ <i>np</i> )	intransitive verb
(( <i>s</i> \ <i>np</i> ) / <i>np</i> )	transitive verb
(( <i>s</i> \ <i>np</i> ) / ( <i>s</i> \ <i>np</i> ))	auxiliary verb
( <i>np</i> / <i>n</i> )	determiner

Table 5.2: Example categories and roles in the Scheme representation

#### 5.2.4 The Initial Lexicon

The final type of input to CLL is the initial lexicon, which is included to bootstrap the learning process. It provides some fairly certain categories for use in the parsing of examples during the learning process, which should in turn constrain the possible categories for other words in the examples. Hence, the aim of this information is to bias the parser in the direction of the right kind of syntactic analyses for the examples. This is shown in Chapter 7 to be, in some respects, psychologically plausible, although provision of a rather broader bootstrapping lexicon with a less well-defined set of categories – perhaps something more like using the slightly annotated corpus described above – might be preferable.

The initial lexicon contains a set of closed-class words with the categories that they may take and a hand-defined probability distribution over the categories. Following the definition of a probabilistic lexicon in Chapter 2, the initial lexicon is defined as a relation of the form

$$(Word, Category, Probability)$$

where a word (*Word*) takes a category (*Category*) with a particular probability (*Probability*). This is represented as a database of facts in Prolog of the form

$$\text{lex}(\text{cc}(Word), Category, Probability).$$

This is, purposely, a format very similar to that used for the learned lexicon, the main difference being that a specific probability distribution is included and the word is marked with the *cc*(...) function. In terms of efficiency, this turns out to be something of a mistake, as the implementation of Prolog being used will no longer effectively index on the first argument of the closed-class lexical entries. Hence, in future, this representation should be modified. Items from the initial closed-class word lexicon are treated differently by the parser in the learning process and it is important that the two are distinguishable, hence the additional function structure around the word indicates that this lexical entry comes from the initial lexicon. An alternative would have been to use a different name for the predicate, which would have had the same effect, but would have undermined the strong link between the initial closed-class lexicon and the learned lexicon.

In some of the experiments no lexicon is used and in others lexicons of varying sizes are supplied. Appendix B contains the different lexicons that have been used and they are described in more detail in the Chapter 9, when the different experiments are described.

## 5.3 The CLL System

In this section the aim is to describe in detail how the system, CLL, uses the inputs described in Section 5.2. To start with, the basic algorithm will be presented to provide an appropriate framework into which the descriptions of all the modules may be placed.

### 5.3.1 The Basic Algorithm

The basic algorithm is shown diagrammatically in Figure 5.1. Two fundamental hypotheses underlie the algorithm.

- syntactic learning and processing occur together;
- learning occurs by choosing analyses leading to the simplest grammar.

The first follows from the observation that children learn and process syntax concurrently (see Chapter 7), and hence it seems appropriate to let each process inform the other.

The second is a version of Occam’s Razor (see for example Li and Vitányi [81]) and as such is the basis of a great deal of work in machine learning.

Perhaps this second hypothesis can more clearly be seen with a simple example. Suppose the two examples below are given to the learner.

```
ex([the, girl, ran]).
ex([the, man, loved, the, girl]).
```

Syntactic analyses with no prior knowledge of the correct categories of the words might allow the CG analyses shown in Figure 5.4. A learner that selects the most general (or most frequent) analysis will select the the left-hand analysis for the first example and the top analysis for the second example, as they contain the most in common (i.e. “the” has the same category in all occurrences, as does “girl”) and so are based on a simpler grammar.

It should be noted that, while the basic algorithm does not use the initial bootstrapping lexicon, it is on examples such as these that it would be very useful. In all the closed-class lexicons the word “the” is assigned the category np/n and would not be allowed to take the category np. Hence, the second parse for each of the examples would not have been allowed.

Each of the sections of the algorithm are described below. In this section I will describe an overview of the approach. Each example in the corpus is taken in turn and has the algorithm applied to it. In this sense CLL is *incremental*.

Firstly, the example is parsed. The parser is supplied with a lexicon built from a number of different resources in different ways depending on the setting of the system, which will be described in the Section 5.3.2.1. The parser is probabilistic using the relative frequency information from the parses of the past examples. Hence the parser returns the  $n$ -best parses



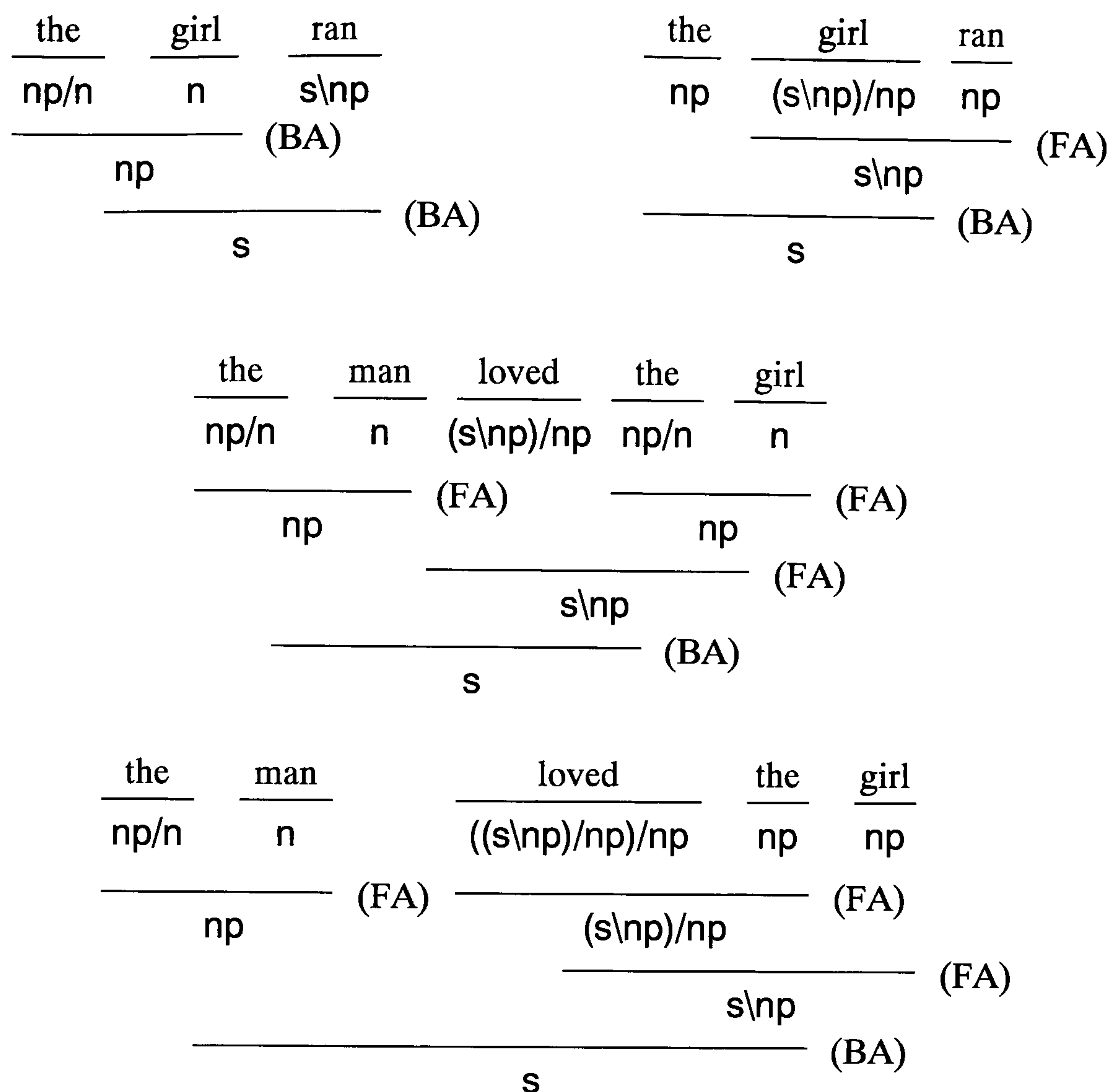


Figure 5.4: Possible parses for examples

of the example that it can produce, where  $n$  is the beam set by the user and best means highest probability.

The advantage of starting with the probabilistic parser is that it immediately generates a set of plausible syntactic analyses. What is already known/hypothesised is used to drive the learning process initially. However, early work suggested that simply selecting the most probable parse did not allow the flexibility required. The best parse focussed too strongly on early decisions for assigning syntactic structure. This suggested a need for a governing principle that could be used for selecting from a set of good possibilities.

As will be clear from earlier discussion, the principle selected was the compression-as-learning approach. In particular, CLL aims to build the smallest possible lexicons that can be used to cover the corpus. The different versions of compression (in effect determining the definition of "smallest" lexicon) are described in Section 5.3.2.2, where the process of selecting the syntactic analysis is described.

In general then, the approach can be considered as using knowledge to determine good options and then using a learning principle to select the best of these options.

Once an option has been selected, the effects of choosing that particular parse are applied to the lexicon that is being learned at the lexicon modification stage (Section 5.3.2.3).

The algorithm terminates when all examples in the corpus have been processed, leaving a lexicon for the corpus and a parsed version of the corpus, which are described in Section 5.4.

### 5.3.2 The Modules

The description above indicates that CLL breaks down into three major modules:

1. the probabilistic parser;
2. the parse selector;
3. the lexicon modifier.

Each of these is described in turn below including their input, output and the knowledge that they use in their processing. This description should provide enough detail to permit both appreciation of the exact techniques used within CLL and the re-implementation of this system.

#### 5.3.2.1 The Probabilistic Parser

The probabilistic parser is central to CLL. It is the embodiment of the intuition that syntactic learning and processing occur together, as it produces likely analyses for examples from which the learner uses compression to select the most likely.

Given the earlier discussion, it should be clear that the parser chosen for CLL will need to be both psycholinguistically plausible and efficient. Efficiency is a driving concern, as the parser is so central to the efficiency of the whole system. The decision on the parsing algorithm is consequently very important and is discussed in a separate chapter (see Chapter 6). Currently, it is sufficient to say that the parser used is a modification of the CKY algorithm (described by Aho and Ullman [3]), which has been adapted to be probabilistic (to some extent following a probabilistic version described by Collins [44]) and to be used with CGs. This approach is considered to be both efficient and psycholinguistically plausible.

In this section, I will concentrate on describing the algorithm and its use in detail.

##### 5.3.2.1.1 The Input

The parser receives three types of input:

1. the example;
2. the categories and rules;
3. the current lexicon.

In fact, these three are used as resources such that the useful information that they can supply is applied at the right moment in parsing. It should also be noted that they may contain different information depending on the setting of CLL, and similarly that the parser may use the information differently depending on its setting.

Taking the simplest setting used in some of the early experiments [134, 135] (see Chapter 9), the example is simply a list of words and there is no initial lexicon (i.e. the lexicon is initially empty). In this case, the example is converted from a Prolog list into a Scheme list, as the parser is written in Scheme (although some of the early work was done with a Prolog chart parser), e.g.

(inventories fell 0.3% in August )



The CG rules present are built into the parser itself (although in a modular way such that rules can be added or removed easily). The categories, however, are not built in. They are instead used, along with the current lexicon, for building a Scheme lexicon which is specific to the current example. Obviously, it is pointless to build a lexicon in Scheme format that contains anything but the entries for the words that are in the sentence, as it would slow the system down. In the earlier system, where a Prolog parser was still being used [134, 135] the full lexicon was available.

In this simple setting, there are two types of word that may need to appear in the lexicon: *unknown* and *known* words. Unknown words are words that appear in the current example, but have not appeared in any of the previous examples, and so do not appear in the lexicon. In contrast, known words have appeared in previous examples and so have entries in the lexicon.

Unknown words are given lexicon entries that assign each of the possible categories (i.e. those in the category database for the given experiment) to the word with a uniform probability distribution, i.e. each word category pair gets a probability of  $1/NC$  where  $NC$  is the number of categories in the category database.

For known words it is slightly more complicated. A word that has been assigned one or more categories already may still be able to take categories that it has not yet been assigned. This is important both to allow a word to take all the categories that it should and also to allow CLL to correct early mistakes (i.e. not to be bound to categories chosen for a word early in the learning process).

To enable this we employ a very simple *smoothing* method. It is assumed that for each category not yet seen for a word, there is a lexical entry for that word-category pair with a frequency of one. A probability distribution can then be calculated for the word in question by first calculating the total frequency of the word (including the extra counts for unseen word-category pairs) and then calculating the relative frequency for all the category word pairs, i.e.  $f/tf$  where  $f$  is the frequency of a particular category word pair and  $tf$  is the total frequency of the word.

A simple example will make clear how this method works in practice. Suppose the system already has the lexicon shown in Figure 5.5. Suppose also, that the set of possible categories is  $\{n, np, s\backslash np, (s\backslash np)/np, np/n\}$ , then given the example:

“John ran”

the lexicon provided for the parser (i.e. the lexicon specific to this example) is constructed in the following way. The word “John” is currently unknown, so it is assigned a uniform probability distribution across all the categories, i.e. there is an entry of  $\frac{1}{5}$  for each category. The word “ran” is already known and has been assigned the intransitive verb category twice. It is assumed that “ran” has been seen with a frequency of one for all other categories. Hence, the total number of “observations” becomes 6. Each unseen category has a probability of  $\frac{1}{6}$ , whereas the intransitive verb category has a probability of  $\frac{2}{6}$ . The lexicon used by the parser is shown in Figure 5.6.

Two points should be noted here. Firstly, it is at this stage that the system moves from using frequencies to using probabilities (albeit based upon relative frequencies). Secondly,

the	np/n	4
man	n	2
girl	n	2
kicked	(s\np)/np	2
ran	s\np	2

Figure 5.5: An example lexicon

John	n	0.20
John	np	0.20
John	s\np	0.20
John	(s\np)/np	0.20
John	np/n	0.20
ran	n	0.17
ran	np	0.17
ran	s\np	0.33
ran	(s\np)/np	0.17
ran	np/n	0.17

Figure 5.6: An example of a lexicon that could be sent to the parser

it should be clear that a well-formed probability model is defined over the lexicon in the sense that the probabilities for a particular word sum to one (noting that the system does not round to only two decimal places, as has been done in Figure 5.6). This, importantly, follows the stochastic CG formalism described in Chapter 2.

While the smoothing system is undoubtedly simple, I would contend that it should be enough for the system to drive it to choose new categories when required. The context of the words will frequently force it to do so, particularly in the first few times of seeing a word. However, this approach remains conservative, in that it makes the assignment of new categories reasonably difficult (they are assigned a low frequency). This stops too much ambiguity being allowed into the system, and ensures that the syntactic analysis (rather than the probabilistic model) drives the generation of new categories for words, which seems an intuitively more likely approach. However, in future it may be interesting to investigate using more complicated methods for smoothing for unknown lexical entries.

There are a number of slight differences that have been applied to this lexicon building process. The simplest is that, in the vast majority of experiments (i.e. all those on the Penn Treebank), some bias towards words being given an np category was included, by giving this category a frequency of 10 rather than the 1 given to other unknown categories. This may be too simplistic, given that once a word has been seen, it will be given a frequency of the number of times it has been used, however, it hopefully provides a small bias for the most common category for unknown words.

The second difference is with the use of the initial bootstrapping closed-class word lexicon. This lexicon has some differences from the standard lexicon. Most importantly, instead of the lexical entry including the frequency, it contains a pre-defined probability. This probability is used by the parser and the frequency information collected in the learning process is ignored. A potential modification could be to allow the frequency information to inform the process,



as this could correct the rather *ad hoc* probability distributions that have been defined for the closed-class lexicons.

The final difference occurs when the NV-annotated corpora are used. In this case, the lexicon generated for the parser is restricted by this annotation. For any annotated word in an examples, the possible categories for that word are restricted to nominal and verbal categories (for the N and the V annotation respectively).

Whatever the setting, the sources of knowledge have been combined to provide the parsing algorithm with the example and a specific lexicon for that example. The rules of CG have also been supplied. These are used within the parsing algorithm itself, which is described next.

**5.3.2.1.2 The Algorithm** CLL is fairly dependent upon the parsing stage of the process, as each example must be parsed with a lexicon which is exceedingly ambiguous (because, apart from the closed-class words, all words can take all categories). The aim then was to build an efficient and psychologically plausible parser. The motivations and decisions regarding the choice of the type of parser are discussed in Chapter 6. In this section, the exact algorithm will be explained in detail.

The early versions of the system [134, 135] used a rather naive implementation of a chart parser in Prolog. However, when it became clear that this would be too inefficient for larger scale experiments, a parser was developed in Scheme (which allowed more control but still had the advantage of fast development) based upon a less naive algorithm.

The basic algorithm was a probabilistic CKY parser for probabilistic Chomsky Normal Form (CNF) context-free grammars, which is described by Collins [44]. This is in turn based upon the standard non-probabilistic CKY algorithm, e.g. [3]. The pseudo-code is reproduced in Figure 5.7 to allow comparison with the modifications described.

The pseudo-code in Figure 5.7 requires some explanation. The chart, or table, is represented by  $p$ . Each of the entries in  $p$  represents the edge of a chart, where the first argument is the starting word, the second the ending word and the final argument the nonterminal symbol for the edge. The value assigned to this function is the probability of the edge. In the initialisation stage, the probabilities of all edges are set to zero.

In the base case, the lexical edges are assigned. Edges are assigned that cover just one word (i.e. start and finish at the same position) if there are rules in the grammar for the word ( $w_i$ ) in that position. In that case, the probability of that rule is assigned to that entry in  $p$ . Note that the CNF rules are represented using the infix operator,  $\rightarrow$ , for the rewrite arrow.

The recursive case is more complicated and is used for joining consecutive edges using an appropriate rule in the grammar. The first loop sets the length of edge being considered, hence all edges covering two words are produced before any covering three edges are added. The second loop sets the start word, and the third loop is used to cycle through the possible intermediate words, i.e. the word at which the first edge to be joined ends and following which the second edge to be joined begins. An edge is added to the chart if its probability (the product of the probabilities of the edges making it up and the rule used for joining them) is greater than any other edge covering the same words with the same nonterminal label. A

```

#initialisation

for all i,j,k
  p[i,j,k] = 0

#base case

for i = 1 ... n
  for k = 1 ... G
    if k -> wi is in grammar
      p[i,j,k] = P(k -> wi)

#recursive case

for s = 2 ... n
  for i = 1 ... n-s+1
    j = i+s-1
    for m = i ... j-1
      for k = 1 ... G
        for k1 = 1 ... G
          for k2 = 1 ... G

            prob = p[i,m,k1] * p[m+1,j,k2] * P(k -> k1 k2)
            if (prob > p[i,j,k])

              p[i,j,k] = prob
              B[i,j,k] = {m,k1,k2}

```

Figure 5.7: Collins [44] pseudo-code algorithm for a CKY parser for PCFGs

back-pointer,  $B$ , is used to store the parse details so that a parse tree can be determined. It should be noted that the particular implementation is not as incremental as a psychologically plausible parser should be, as the smaller arcs across the whole sentence are built before the larger ones (see Chapter 6), although it still remains relatively incremental and, for the sake of efficiency, this implementation will be used, although in future a more incremental system would fit better with an aim to be psychological plausible.

This algorithm is fairly efficient with respect to natural language parsing, as it is cubic in time with respect to the length of the input string and the number of nonterminals, and it is quadratic in terms of space with respect to the length of the input string. However, the reconstruction of parses can cause problems. Where there is only one parse this is not significant, but where there can be exponentially many parses, as in general with NL parsing, then the parser is still exponential (in both time and space) if one wants to recover all parses. It is also not a very psychologically plausible approach to parsing to construct the parses after all of the sentence has been processed (see Chapter 6).

The algorithm is, in many senses, very appropriate for CG parsing, as the binary branching nature of CG derivations makes them very similar to the CNF grammars. However, some simple modifications to the recursive case are necessary to make it appropriate for CG. Essentially, instead of the grammar having a large set of rules that are searched for right-



hand sides that match with edges in the chart, a large number of categories are searched to instantiate a small set of general rules.

The parser also needs to be changed to build the  $n$  most-probable parses. This is a matter of allowing up to  $n$  edges for the same root category of the subtree covering the same words.

Given both psychological plausibility and the need to extract a set of parses, the derivations are stored along with the edges. In future, for efficiency, it may be wise to implement the back-pointer system for recovering parses, especially as the experiments tend to use a small beam and so the parse recovery process would probably be more efficient than storing the parse trees with the edges.

Finally, in an earlier version of the system [134, 135, 136] there was one constraint for removing very unlikely edges. This, in principle, removes the certainty of getting the most probable edges, but in practice it is very unlikely to do so and seems psychologically plausible as it stops the further consideration of very improbable intermediate edges. The constraint simply removes (or does not add) an edge that is significantly less probable than another edge on the chart covering the same words. This constraint was dropped as it was no longer needed to make the parser efficient enough to run.

Figure 5.8 shows the modified algorithm for the probabilistic  $n$ -best CG CKY parser. The algorithm is of the same form as the first version (Figure 5.7). The main difference is that  $p$  now stores pairs containing the parse and probability information for each edge. The other difference is that the rules have been made explicit, so that each iteration checks for all uses of each of the functional application rules. However, the results of these rules are only added to the chart ( $p$ ) if they are part of the current  $n$ -best edges. The noun-phrase-composition rule, which is used in some experiments is included, but there are obviously some cases where this is not required. This rule allows construction of complex noun phrases in a way that avoids some of the intricacies of a method within a more pure CG context, i.e. the actual adjunct/complement structure does not need to be determined. In future, a more intelligent method of building complex noun phrases needs to be investigated. The current approach is too naive in looking to apply the rule whenever it is possible. This is costly, as all words (apart from the those in an initial lexicon) will be hypothesised as being an np category, which leads to a proliferation of np edges. An approach that applied this rule lazily, only when it was needed to complete a parse, would probably significantly improve efficiency.

It should be noted that the algorithm maintains the notation of the previous algorithm. Again  $p$  is used as the chart structure with three arguments indicating the start and finish point of the edge and the category of the edge. The value of this function is now a pair of the parse subtree and the probability of that tree. The value of  $b$  is the value of the beam set by the user. A case is provided for each rule and the base case assigns lexical entries to the words.

The algorithm seems to maintain a good balance between efficiency, psychological plausibility and appropriateness for CG parsing. It has been used for a large range of experiments and has thus far been effective enough. However, in the future, it will almost certainly be necessary to improve the efficiency for work on larger sentences and to include the extra

```

#initialisation

for all i,j,k
  p[i,j,k] = ([],0)

#base case

for i = 1 ... n
  for k = 1 ... G
    if lex(wi,k) is in grammar
      p[i,j,k] = ([wi, k],P(lex(wi,k)))

#recursive case

for s = 2 ... n
  for i = 1 ... n-s+1
    j = i+s-1
    for m = i ... j-1

# forward application

    for all p[i,m,k/k1] and p[m,j,k1]

    prob = P(p[i,m,k/k1]) * P(p[m+1,j,k1])
    if (prob > edge p[i,j,k] with minimum P value)
      or (|p[i,j,k]| < b)
        minimum P value p[i,j,k] = ([k, [k/k1, k1]], prob)

# backward application

for all p[i,m,k1] and p[m,j,k\k1]

    prob = P(p[i,m,k1]) * P(p[m+1,j,k\k1])
    if (prob > edge p[i,j,k] with minimum P value)
      or (|p[i,j,k]| < b)
        minimum P value p[i,j,k] = ([k, [k1, k\k1]], prob)

# np composition

for all p[i,m,np] and p[m,j,np]

    prob = P(p[i,m,np]) * P(p[m+1,j,np])
    if (prob > edge p[i,j,np] with minimum P value)
      or (|p[i,j,k]| < b)
        minimum P value p[i,j,np] = ([np, [np, np]], prob)

```

Figure 5.8: The modified CKY parsing algorithm used by CLL



rules needed to handle movement.

**5.3.2.1.3 The Output** The parser returns a list of the  $n$  (or fewer if less than  $n$  exist) parses, which consist of the syntactic structure and the probability of that structure. The Scheme parser saves this to a file in Prolog format, e.g.

```
ps([([s, [[np, 'inventories'], [bs(s, np), [[bs(s, np), [[fs(bs(s,
np), np), 'fell'], [np, '0.3_%']]]], [bs(bs(s, np), bs(s, np)),
[[fs(bs(bs(s, np), bs(s, np)), np), 'in'], [np, 'august']]]]]]],
6.64353094777256e-5])).
```

The list structure is used to indicate the bracketing and allows an easy translation from the Scheme list structure produced for the parse. It also allows a clear distinction between the functional representation of the categories and the parse bracketing. This makes it a better choice than the other option, which would be to use some kind of functional structure. The output from the parser is then picked up by the learner and used in the next stage.

Where no parse has been found, an empty list is returned and this is interpreted by the system as a failure, at which point CLL moves on to the next example, as the rest of the stages of the learning process cannot be performed.

### 5.3.2.2 The Parse Selector

Having obtained a set of the most probable parses based upon the current hypothesis of the lexicon, CLL must then determine which of the parses to select. This parse will then become the current annotation for that example in the corpus. It will also form the input to the lexicon, i.e. the lexical entries required for the parse will be added to make a new hypothesis for the lexicon. Hence, it is this module that makes the specific choice as to which parses will be used in the building of the lexicon and the annotated corpus. A graphical summary of the parse selector is provided in Figure 5.9.

The intuition behind this stage is that, while the current hypothesis will be of some use in determining the best parse, an external principle will be needed to balance this in terms of defining the type of lexicon built. The notion of compression-as-learning, which has been applied to natural language learning in a number of situations (e.g. Wolff [141], Osborne [95] and Osborne and Briscoe [96]), has been selected as both a computationally effective methodology, as well as showing some psychological plausibility. As mentioned, there is some evidence that it can be used in a linguistically plausible method. However, part of the hypothesis being investigated is that applying compression to build large-scale CG lexicons allows us to learn useful lexicons effectively.

There are various ways of applying compression to the lexicon-learning problem that has been defined. The two most obvious things to compress are the lexicon and the lexicon taking frequency (i.e. the data) into account and so experiments have been performed with two metrics: one compressing the lexicon and the other compressing the lexicon and the corpus. In future, it would be useful to look at other metrics, for example using an MDL approach by compressing both corpus and lexicon separately.

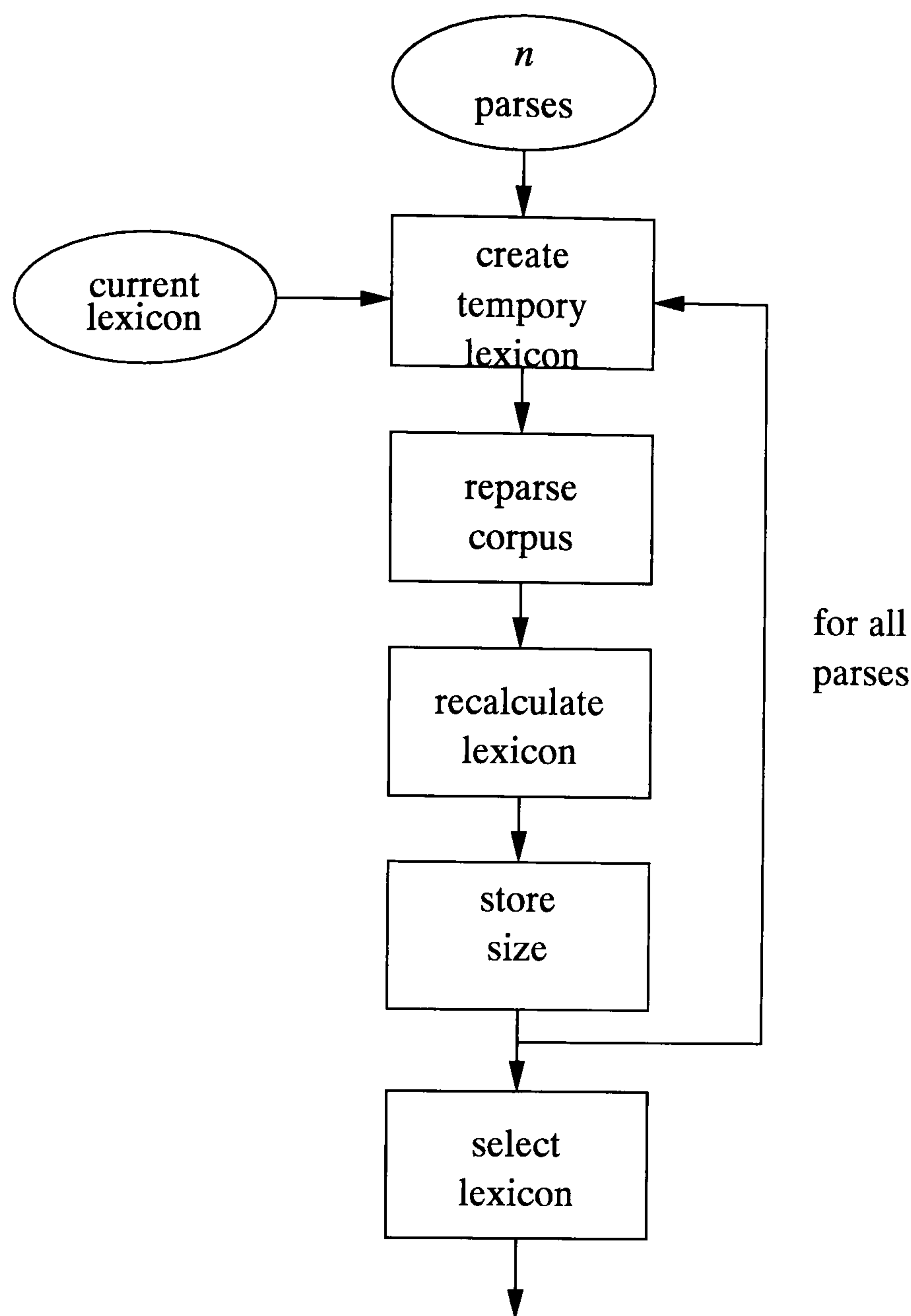


Figure 5.9: A diagrammatic representation of the parse selection module

In both cases, it is necessary to define the notion of size that will be used by the system in selecting the “smallest”, i.e. the most compressive lexicon. The size of the lexicon (without frequency information) is easy to define in the context of the lexicons used by CLL. It is simply the number of lexical entries (i.e. the number of word, category and probability triples). This can be calculated by counting the number of Prolog facts in the lexicon.

The size measure for the lexicon including frequency information is more complicated. The compression metric used in CLL is based upon the encoding scheme of Osborne [95] and Osborne and Briscoe [96], where a Minimum-Length Encoding is calculated for the corpus (the compression metric used does not follow their MDL approach, as discussed in the previous chapter). Although in Osborne [95] the encoding is based upon category rather than lexical entry. The compression metric is considered in information-theoretic terms, where the aim is to minimise the size of the of the corpus annotated with the categories for the words. This size can also be considered as the size of the lexicon taking the frequency of the occurrence of lexical entries into account. The *encoding* probability of a lexical entry,  $l_i$  is defined as the relative frequency of the lexical entry with respect to the total frequency of lexical entries, i.e.

$$P(l_i) = \frac{f(l_i)}{\sum_{\forall l \in L} f(l)}$$



where  $f(l)$  is the frequency of  $l$  and  $L$  is the lexicon (the set of lexical entries). Due to the equivalence between the probability of an item and the number of bits required to transmit it, the Shannon Complexity can be used to calculate the minimum number of bits required, i.e., given an item  $l$  with a probability  $P(l)$ , the Shannon Complexity is:

$$-\log_2(P(l)).$$

Hence, the size of the lexicon, taking frequency into account, is

$$\sum_{\forall l \in L} -\log_2(P(l)).$$

Having determined the size measures that the system will attempt to minimise, one must discuss the algorithm that the system uses to select the parse that leads to the most compressive lexicon.

One possibility would be to add the lexical entries used in a parse to the current lexicon, i.e. where a lexical entry from the current lexicon has been used, then add one to the frequency and, where a new lexical entry has been used, add this to the lexicon with a frequency of one. It becomes a matter of selecting which parse provides the minimum addition to the size of the current lexicon.

There is, however, one obvious problem with this approach. Early mistakes in parse selection (when there is very little evidence to guide the selection process) are retained by the system. This has three bad results. Firstly, the system retains mistaken lexical entries, which, although hopefully of low probability, do not improve the quality of the lexicon. Secondly, the system retains unlikely parses for examples which would not be assigned by the final lexicon, thus making the annotation of the corpus much less useful. Finally, the system continues to use unlikely early lexical entries to inform the learning process, which will reduce the effectiveness of the learner.

Given this analysis, a better approach would seem to be to review earlier analyses and try to correct mistakes and deduce a more consistent lexicon and parse annotation. This is perhaps not a psychologically plausible approach, as it is not common to allow that a child stores learned parse information [14]. However, as a computational approximation it is not unreasonable, especially as it allows us to achieve the goal of obtaining two useful resources (a CG annotated corpus and a large scale CG lexicon). One can consider the reparsing as a way of approximating large amounts of data. Instead of seeing words and phrases many times in a large of data, a smaller amount of data is used, but the words and phrases are re-used.

In future, it would be interesting to pursue a simple thresholding scheme that removed unlikely lexical entries. This would possibly be more efficient and more psychologically plausible, but would not allow such effective re-evaluation of hypotheses.

The exact approach taken is to build a modified lexicon for each of the  $n$  most-probable parses hypothesised by the method above of adding the lexical information from the parse to the current lexicon. This lexicon is then used to reparse all the examples seen so far. The most probable parse with this approach is then used to build a alternative lexicon (using the

lexical assignments from each parse) and the size of this lexicon is calculated. This is done for all the parses and the one that results in the smallest alternative lexicon is chosen as the best parse.

There are a number of advantages to this approach. Firstly it does allow early mistakes to be corrected, and so a compressive lexicon is actually built and an up-to-date annotation is maintained. Perhaps most importantly, it allows CLL to focus more rapidly in its learning procedure, as early erroneous decisions are generally removed rather than continuing to inform the learning process.

There are some issues to be noted however. As already mentioned, the use of stored parse information is probably implausible. However, it can also be seen to be inefficient – after every example is parsed, all previous examples must be parsed. Finally, it does not ensure the *most* compressive lexicon, as the approach is iterative. The new lexicon built could in turn be applied to all the examples again to see if any of the parses change and thus a new lexicon could be built, which could in turn be applied to the previous examples etc.

The second issue is a matter of algorithm efficiency. It should be clear why it was noted earlier that the efficiency of the parser was important. However, a number of steps can be taken to reduce the amount of parsing. The most obvious of these is to reparses only those examples which might change. Hence, examples without any of the same words as the current example are not reparsed. Similarly, sentences that do have the same words, but with the same categories as the current example's parse are not reparsed. Only examples with the same word, but a different category, are reparsed. Only reparsing examples that might change, significantly reduces the amount of processing performed by CLL.

In earlier systems [134, 135, 136], the parse-selection stage used previous parses for filtering, but this causes problems with sentences which could take a variety of other parses given the new lexicon, hence this is no longer used in the system.

The final issue, iterative compressing, is one that could be investigated further in future. Currently we have one stage of reparsing. This maintains respectable empirical efficiency and appears to compress effectively (perhaps too effectively – see Chapter 11). However, there may be some call to investigate this compression further in future.

At the end of the parse-selection stage, a set of sizes (defined by the compression metric being used) have been stored along with the the appropriate parses of the current example. The smallest of these is used to select the correct parse.

### 5.3.2.3 The Lexicon Modifier

Having selected the parse for the current example, that parse is used to generate that lexicon from the previous examples and itself, and this lexicon is set as the current lexicon. The most probable parses produced using this lexicon are selected as the current annotation for the examples seen thus far. The lexicon is stored as a set of facts similar to those in the initial lexicon, but without the closed-class marking, e.g.

```
lex(merrill_lynch_ready_assets_trust, np, 4).
```

Hence, at this stage, the system has a most up-to-date lexicon and parse annotation. If there are more examples to be processed, then the system returns to the beginning of the



process and starts with the next example. If not, the system halts.

### 5.3.3 Properties of the Algorithm

It is common to discuss the theoretical properties of algorithms to determine how effective these will be and whether or not they will scale up. In this section, two key areas will be discussed:

- complexity;
- convergence.

It is probably worth noting that the system is not designed to be a theoretically pure machine-learning algorithm. It will be remembered that this was not one of the aims outlined in Chapter 1. Instead, the system is designed to work on practical problems and produce results that could be useful (if the system works well). Hence, the algorithm has currently taken some heuristic shortcuts to achieve this.

#### 5.3.3.1 Complexity

The complexity of CLL essentially rests upon the parser. In the worst case, for each example given to the learner, it is parsed and all the previous examples are parsed. If the number of examples is  $E$ , then the number of times the parser is called has an upper bound of the sum of the following arithmetic series.

$$1, 2, 3, \dots, (E - 1), E$$

Using the standard result for the sum of an arithmetic series:

$$\sum_{k=1}^n (a + bk) = \frac{1}{2}(n^2 + 2an - bn),$$

where  $a$  is the first term,  $b$  is the common difference and  $k$  is the number of times the common difference has been added, it is simple to calculate the upper bound, which is:

$$\frac{E^2 + E}{2}$$

This means that the number of times the parser is called is bound by a quadratic in the number of examples.

This shows only the number of times the parser is called. It is the parser itself that remains the key. A CKY parser is in general cubic in time and quadratic in space requirements. However, as already mentioned, these bounds are achieved by not actually returning the parses, but enough information to recover the parses. Given that there can be an exponential number of parses (in the length of the string) it is still the case that to retrieve all the parses requires time that is exponential in the number of words.

The probabilistic CKY algorithm shown in Figure 5.8 would be exponential in the number of words in the example with respect to both space and time because the parse information

is stored on the edges. However, this is restricted by the use of the beam and a finite set of categories which can be used. Given a beam  $b$ , an example of length  $n$  and a set of categories  $C$ , then it is possible to calculate the bound of the number of edges with respect to these values.

The maximum number of edges covering any sequence of 1 or more words is the same as the beam multiplied by the total number of categories in the database, i.e.  $b \times |C|$ . The total number of sequences of words is the same as the number of pairs that can be picked from the sequence  $1, 2, \dots, n$  such that the first number in the pair is less than or equal to the second number. (If the numbers are equal, then, given the algorithm presented, the edge would cover one word). In this case, there are  $n$  pairs where the first number is one,  $n - 1$  pairs where the first number is 2 and so on until there is only 1 pair when the number is  $n$ . This can again be reversed and phrased as an arithmetic series:  $1, 2, \dots, n - 1, n$ . Using the formula for the sum of arithmetic series where  $a = 1$ ,  $b = 1$ , then the total number of sequences of words is:

$$\frac{n^2 + n}{2}$$

Hence, in the worst case, the number of edges, which bounds processing space is

$$b \times |C| \times \frac{(n^2 + n)}{2}$$

and so is quadratic, even if it may potentially be quite large. This bound on the number of edges maintains cubic time complexity of the CKY algorithm (due to the three nested loops of the algorithm).

Hence, the number of times the parser is called is bound by a quadratic in the number of examples in the corpus, and the parser itself calculates a number of edges that is bound by a quadratic in the length of the example. Although these bounds are high, in practice they have remained within reasonable computational complexity, i.e. such algorithms can be used, but may take some time. In fact, this worst case bound will seldom be met, as not all the previous examples are reparsed, but only those that might be affected and as it will never be the case that all sequences of words can take all categories (the restrictions of providing legitimate parses will prevent this, as will the initial lexicons and the use noun and verb marking).

Practically, it has been shown that CLL has some problems in completing large-scale experiments. However, this is due to time constraints (experiments had to be completed reasonably fast to get an appropriate range) and hardware and software constraints (improvements in both the implementation and the hardware the system is run on has shown significant performance improvements).

### 5.3.3.2 Convergence

It is common for machine-learning algorithms to have some provable idea of convergence, i.e. the algorithm can be shown to settle upon a grammar that no further examples from the language will change.

In the context of a system that aims to be computationally practical, this kind of conver-



gence result is probably unnecessary. Humans continue to modify their language throughout their life. Hence, a human learner does not appear to ever completely converge upon a grammar. While this may, in part, be due to a changing language environment, it is not certain that convergence need be a strict requirement. However, the learner should be seen to be moving in the direction of a fairly settled grammar. The CLL system can be seen to be doing this by producing a compressed lexicon/corpus. If the metric were to produce the most compressive lexicon, e.g. using the iterative version of the algorithm described above, then the learner could be shown to converge, as the most compressive lexicon for a fixed language is fixed.

However, such an approach is computationally expensive and so a simple restriction is placed upon the algorithm so that only one iteration of previous examples is used. The algorithm is not certain to converge upon the most compressive lexicon, or on the correct probability distribution of lexical entries. It will, however, provide a reasonable approximation.

Given our aims, this would seem to be the ideal kind of algorithm. While further work on convergence issues may be of some theoretical interest with respect to a theoretical version of the algorithm, it will not be of any practical interest at this stage, especially as the work here is concerned with computational practicality for a syntax-learning system.

## 5.4 The Output

It should be clear from the discussion of CLL given above that there are two main elements of the output:

- a large-scale Categorical Grammar lexicon;
- a Categorical Grammar annotated corpus.

It is also possible to obtain all the other possible parses that have been considered for the examples in the corpus, but as yet these have not been used.

The format of these resources is the format used within the system. Hence, the lexicon is a large set of Prolog facts of the form of the lexical entries described in Section 5.3. Similarly, the parsed examples are of the format shown in Section 5.3.2.1.3, although they are not stored with probabilities and have the example number attached, e.g.

```
ex(24, [s, [[np, i], [bs(s, np), [[fs(bs(s, np), np), saw], [np, neither]]]]]).
```

Both these resources could, in principle, be very useful for various Natural Language Processing tasks, including parsing, training parsers and perhaps bootstrapping supervised learning systems.

## 5.5 A Worked Example

The description of the mechanisms of CLL given above is detailed, but perhaps will be made clearer with a worked example. This will obviously be a toy example, but it will suffice to make the inner workings of the system clearer. The example shown will assume

the unsupervised setting with an initial lexicon. Compression will be based purely on the number of lexical entries. However, it should be reasonably clear where the other settings will differ and how the approach can easily be modified. Initially, assume the system is being used with an initial lexicon that contains the lexical entry:

`lex(cc(the), fs(np,n), 1)`

for the word “the”.

Suppose the examples processed so far are:

“the donkey kicked John”

“John kicked the donkey”

The parses that have been assigned thus far are shown in Figure 5.10. These maintain the restriction of the lexical assignment for “the”, but still allow an incorrect parse for the first example. The current lexicon can now be derived and is shown in Figure 5.11.

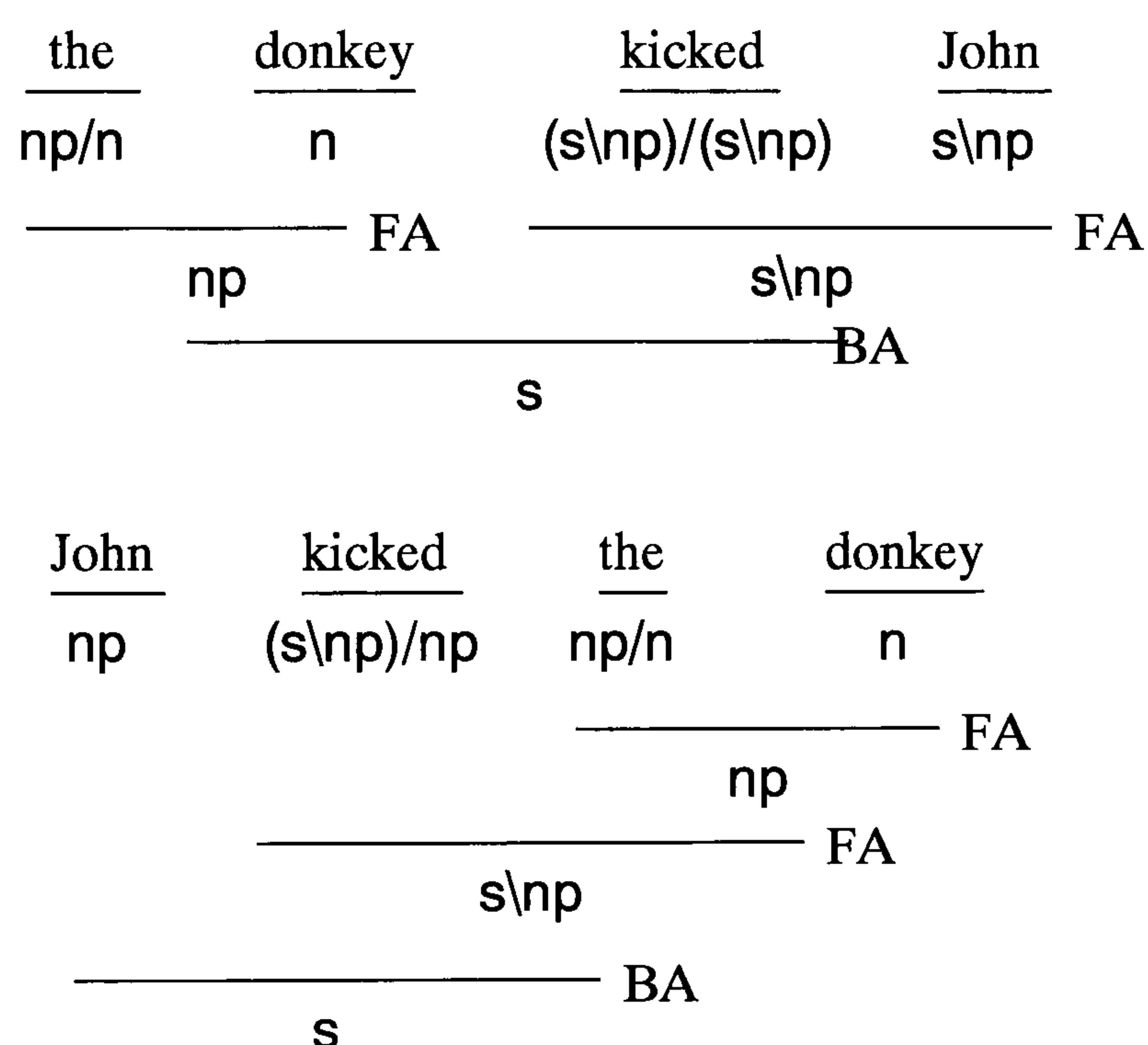


Figure 5.10: Initial analyses of the examples that have already been processed

```
lex(the, fs(np, n), 2).
lex(donkey, n, 2).
lex(kicked, fs(bs(s, np), bs(s, np)), 1).
lex(kicked, fs(bs(s, np), np), 1).
lex(john, np, 1).
lex(john, bs(s, np), 1).
```

Figure 5.11: The current state of the lexicon in the worked example

Suppose now that the next example is:

“Mary kicked the big donkey”

Suppose also that the beam is set to two. The two parses for this example given by the system are shown in Figure 5.12. If it is assumed that the category database contains a set



of ten categories, then the probabilities shown can easily be calculated. The probability for “Mary” being an np is taken from its frequency, 1, and then smoothed for all possible other categories, i.e. nine other categories given a frequency of 1, which gives it the probability of 0.1. Similarly, “kicked” is given the probability 0.1 for taking the category (s\np)/np. Note that there is no parse that allows it to take the auxiliary verb category which is available in the lexicon, even though this is just as probable. This shows how the symbolic and probabilistic constraints combine to form a good bias. The word “the” has a fixed category and probability from the initial lexicon. The word “big” has not been seen before and so is assigned a set of smoothed probabilities. Given a set of ten categories, it is assigned a frequency of 1 for all the categories apart from np for which it is assigned a frequency of 10. Hence, the total frequency is 19 and the relative frequencies of the categories n/n (parse 1), or n (parse 2) are both 0.05. The probabilities of the two categories for donkey are calculated similarly. The category n has a frequency of 2 in the lexicon and all other categories are given a frequency of 1, hence the relative frequency of category n is 0.18 and the relative frequency of np\np is 0.09. The overall probabilities are shown, but once these two parses have been selected as the two most probable, their probabilities are ignored.

Parse 1	<u>Mary</u>	<u>kicked</u>	<u>the</u>	<u>big</u>	<u>donkey</u>	
	np 0.1	(s\np)/np 0.1	np/n 1	n/n 0.05	n 0.18	
						FA
					n 0.009	
						FA
					np 0.009	
						FA
					s\np 0.0009	
						BA
					s 0.00009	
Parse 2	<u>Mary</u>	<u>kicked</u>	<u>the</u>	<u>big</u>	<u>donkey</u>	
	np 0.1	(s\np)/np 0.1	np/n 1	n 0.05	np\np 0.09	
						FA
					np 0.005	
						BA
					np 0.00045	
						FA
					s\np 0.000045	
						FA
					s 0.0000045	

Figure 5.12: The hypothesised parses for the next example

The two parses are now used in the parse-selection stage. This returns two hypothesised lexicons, built by adding the lexical entries used in the parses to the current lexicon. These are shown in Figures 5.13 and 5.14 for parse 1 and parse 2 respectively.

Both lexicons are then used to parse the previous examples with words in common, but with different categories. In this case, the word-category pairs in parse 1 differ from the word-category pairs in the parse of the first example with respect to the category assigned to “kicked”, but do not differ from the word-category pairs of the parse of the second example, so only the first example will be reparsed. Here the parse actually changes to that shown

in Figure 5.15, which is a correction of an early mistake. This produces the lexicon in Figure 5.16.

In the second case, both previous examples need to be reparsed because the category `np\np` has been assigned to the word “donkey” in parse 2. However, because the category of “the” is set, no more probable parses can be calculated using these parses. Hence, the lexicon remains the same as that shown in Figure 5.14. With respect to the number of lexical entries, it is clear that the smallest lexicon is that shown in Figure 5.16, which means that this is now set to the current lexicon, and the current annotation is the parse shown for the first example in Figure 5.15, for the second example in Figure 5.10, and, for the current example, it is the first parse in Figure 5.12. These also turn out to be the correct parses and show how CLL can correct earlier mistakes when it is supplied with more data.

Have completed processing one example, the system would now return to the start of the process and work with a new example from the corpus.

## 5.6 Conclusions

This chapter has provided a detailed explanation of CLL that both aids the understanding of the the workings of the system and also aids the understanding of the intuitions behind the system and how they have been implemented. While there are a number of possibilities for modifying the algorithm, the current system has been well motivated and is suitable for a very wide range of experiments. In this chapter, I have also shown that CLL is an implementation of the computational model developed in Chapter 3.



```

lex(the, fs(np, n), 3).
lex(donkey, n, 3).
lex(kicked, fs(bs(s, np), bs(s, np)), 1).
lex(kicked, fs(bs(s, np), np), 2).
lex(john, np, 1).
lex(john, bs(s, np), 1).
lex(mary, np, 2).
lex(big, fs(n, n), 1).

```

Figure 5.13: Hypothesised lexicon 1 resulting from parse 1

```

lex(the, fs(np, n), 3).
lex(donkey, n, 2).
lex(donkey, bs(np, np), 1).
lex(kicked, fs(bs(s, np), bs(s, np)), 1).
lex(kicked, fs(bs(s, np), np), 2).
lex(john, np, 1).
lex(john, bs(s, np), 1).
lex(mary, np, 2).
lex(big, n, 1).

```

Figure 5.14: Hypothesised lexicon 2 resulting from parse 2

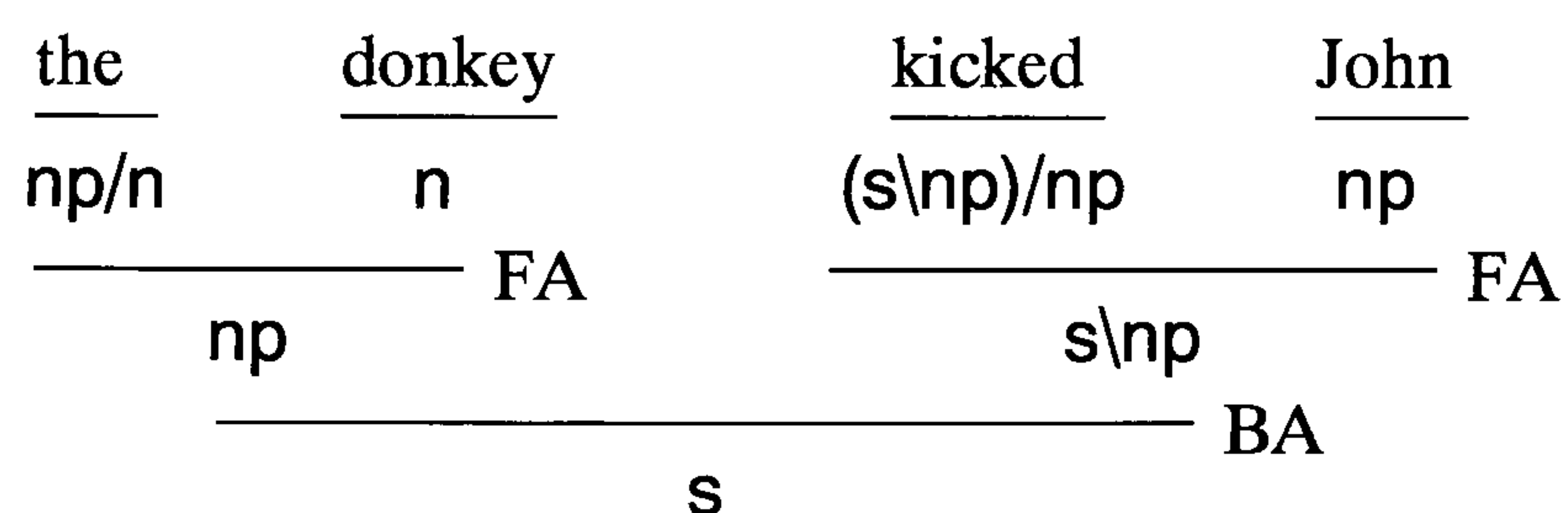


Figure 5.15: The changed parse when reparsing with the first hypothesised lexicon

```

lex(the, fs(np, n), 3).
lex(donkey, n, 3).
lex(kicked, fs(bs(s, np), np), 3).
lex(john, np, 2).
lex(mary, np, 2).
lex(big, fs(n, n), 1).

```

Figure 5.16: Modified lexicon 1

# Chapter 6

## The Parser

In this chapter, I present the motivations for the selection of the  $n$ -best probabilistic-CKY parser, which is used in CLL and was described in Chapter 5. The centrality of the parser in CLL, both practically and theoretically is discussed in Section 6.1, motivating the remaining discussion of parsing in this chapter.

Given that the parser is such an important element of CLL, selecting a parsing algorithm from the many available is a crucial decision in the design of the learner. A set of requirements for the parser are presented in Section 6.2. These requirements are then discussed with respect to various general parsing techniques to determine the right kind of parsing approach for the problem (see Section 6.3).

It will be clear from the description of CLL in Chapter 5, that the parser is required to have some mechanism for selecting a set of the “best” parses. In Section 6.4, some selection methods are discussed and the relationship between the stochastic CG being used and the notion of “best” is stated.

Finally, having presented the motivations for the CKY algorithm, I draw some conclusions and discuss some possibilities for improving and extending the parser in Section 6.5.

### 6.1 The Centrality of the Parser

One of the aims when designing the CLL system was to combine stochastic and symbolic constraints on the search space so that the learner converges upon the correct grammar. The parser can be seen as the fulcrum of this principle within CLL, as it is where the symbolic and the statistical constraints meet. The symbolic constraints, i.e. the CG grammar rules and categories, define the set of legal parses for an utterance. The parses define which lexical assignments can be used and so define the space of lexicons that can be hypothesised by CLL. Hence, the parser is at the centre of the symbolic constraints placed upon the search space. Similarly, the stochastic CG is used to define which are the “best” analyses and so which will go forward as possibilities for the input to the lexicon. So, the parser is at the centre of the statistical constraints upon the search space.

Another of the aims was to combine syntax learning and processing, which would appear to be a practical way for achieving hypotheses for the lexicon and would be similar to the human approach [14]. The parser is central to this aim, as it is both the mechanism



for syntactic processing, and the primary source of information for updating the lexicon. Therefore, the parser can be seen as a central factor in meeting these theoretical aims.

There is, however, another sense in which the parsing is central to the system. The outline of the implementation of the learning model, including the complexity analysis (Chapter 5), shows that a significant part of the processing of the system is parsing. Each example is probabilistically parsed when it is first encountered and then, for each parse of each new example, the parser is applied to a, potentially, large number of the previously encountered examples to determine if the most probable parse that they are annotated with should change. In fact, the majority of the work performed by the system will be parsing and so, very practically, the parser is central to the effectiveness of CLL.

Hence, it can be seen that parsing is central both to the theoretical basis and to the practical implementation of the system. The centrality of the issue justifies the existence of this chapter for discussing the most appropriate parsing approach to follow and its implementation. The most obvious next step is to consider the fundamental requirements for a parser for CLL.

## 6.2 The Requirements for the Parser

Three general requirements are applied to the decisions made with respect to the parsing algorithms. These are, in descending order of importance:

- computational effectiveness,
- linguistic appropriateness,
- psychological plausibility.

The implications of each of these general requirements are discussed with respect to the parser below.

### 6.2.1 The Computational Effectiveness of the Parser

An issue related to both the psychological plausibility and the computational effectiveness, is the efficiency of the parser. It is especially clear that the parser must be fast. This is not only because of the apparent speed of humans in parsing sentences. In the context of this system, parsing is central, as discussed above. In fact, the majority of the processing is parsing. Hence, a slow parser would render the system useless for anything but the simplest of corpora.

Experience has shown that the performance of the system has had to be consistently improved as more complex corpora have been used and that the main area for improvement has been that of the parser. Small improvements have made a significant difference to run times, because of the weight that the system places upon the parser.

A further computational requirement with respect to CLL, is that the parser needs to be able to generate a set of the  $n$ -best parses. In other words there must be some computational mechanism for returning a set number of parses and for ranking the quality of parses to ensure that those returned are the best.

### 6.2.2 The Linguistic Appropriateness of the Parser

It may seem rather obvious to state that the parser needs to be designed to use a CG given the discussion in the previous chapters. However, the choice of grammatical formalism does have implications on the kind of algorithm that needs to be applied.

The most obvious limitation of using a lexicalised formalism like CG is that most of the information for building the correct parses is contained within the lexicon. Hence, the most efficient approach will immediately use this information to constrain the parsing.

Secondly, given that the CG defined in Chapter 2 is based entirely on binary combination rules, it may be possible to take advantage of algorithms that have been shown to be efficient for binary-branching grammars. Extensions of the grammar may change this current restriction, as there is potential for adding unary and ternary rules. However, currently, the binary-branching constraint remains useful.

Finally, the parser is not simply attempting to find the right bracketing of the sentence, or the correct logical relationships between the words. Nor is a partial parse a valid solution. The aim is to build full parses providing both a bracketing of the sentence and an appropriate labelling at each level. A complete parse should provide most of the information for building the lexicons, ensuring that they contain linguistically plausible entries for building parses that are structurally accurate and labelled usefully.

### 6.2.3 The Psychological Plausibility of the Parser

According to Crocker [46], there are two issues in psycholinguistic theory that relate to making decisions about the sort of parsing algorithm to use.

- Incrementality
- Processing limitations

Incrementality insists that the words are given to the system in the order they are spoken (or read) and they are attached to the output parse before the next word is received, i.e. both the input and the output are processed incrementally. Processing limitations deal with the limitations of human processing time and human memory size, which cause certain phenomena in sentence processing.

A parser should be subject to these psycholinguistic issues with one possible caveat, which is that, in the case of a learner, the parser may be considered to not be fully formed yet, as there is still syntactic knowledge to be learned. Hence, the aim is to select an algorithm that is reasonably incremental and could be extended, with further knowledge, to be more so. Similarly, the algorithm should be able to take processing limitations into account. Hence, the algorithm should be efficient, but will also have problems with the same types of examples that humans have problems with.

## 6.3 General Parsing Approaches

In this section, some of the general approaches to parsing are reviewed with respect to the requirements above.



### 6.3.1 Parse Order

The order in which the words of an utterance are parsed is one of the dimensions upon which parsers can vary. The most common approach is probably left-to-right, i.e. in the same order in which the words are spoken (rather than how they are written on a page). The obvious alternative is right-to-left. However, there are approaches that deal with utterances in more linguistically motivated orders, e.g. head-driven parsing [126, 44], where the head sibling in the parse tree is isolated and used to drive the parsing process.

From a psychological perspective, which requires incremental processing, a left-to-right parser is the only possibility. With respect to linguistic plausibility, the head-driven approaches are elegant, as they use linguistic knowledge to produce better parses. However, at the learning stage it will probably not be possible to identify the head very easily. The right-to-left approaches can prove advantageous computationally for handling movement. For example, movement could be identified and the rules to handle moved elements could be applied when the verb in a sentence is reached (from the right) before all the complements it requires have been handled. Movement rules would then only be applied lazily, when it is discovered that they are needed. This approach to parsing sentences including movement is one possibility under discussion for extending the current parser, so CLL can be applied to corpora including examples with movement.

Although there are advantages to each approach, it would seem entirely acceptable that a left-to-right parser can be used that will be computationally efficient, produce good parses (if not the best) and will maintain psychological plausibility. Hence, a left-to-right parser will be used.

### 6.3.2 Top-Down and Bottom-Up Parsing

In top-down parsing, the parser starts with the grammar rules, usually some initial symbol (e.g. the S symbol in a phrase structure grammar) and rewrites the symbol until a structure including all the words of the example is produced. Simple top-down algorithms are described by Gazdar and Mellish [57], Allen [4] and Crocker [46] (who also provides evaluation with respect to the psychological plausibility).

Bottom-up parsers, instead of starting with the grammar and working down to the words, work the opposite way. They start with the words in the examples and the categories that they may belong to. The grammar rules are then used to build syntactic structure to join the words until the entire example is covered by one category (usually the sentence category). Again there are a variety of sources for basic bottom-up parsing algorithms [4, 57] and Crocker [46] provides an analysis of such algorithms with respect to the psychological issues.

In terms of linguistic appropriateness, top-down algorithms are difficult to use with CG. Instead of starting with the constraints available in the lexicon, the top-down parser would have to hypothesise the intermediate parse structure, including a set of new intermediate categories, before the lexical entries can be included in the analysis. Bottom-up parsers are therefore more sensible with a lexicalised grammar like CG, as they begin with the lexical entries, which contain all the constraints for building the correct parses.

With respect to psychological plausibility, top-down parsers can be designed to achieve

complete incrementality. As each word is received it can be attached to the complete analysis so far. However, with bottom-up parsers, the structure above the lexical items is commonly not generally constructed until further words have been received, i.e. the input is received incrementally and structure is built incrementally, but output structure can remain in many parts.

A simple example of this difference would be the parsing of the verb phrase “kicked the ball”. A top-down algorithm would probably follow the sequence shown in Figure 6.1, where the verb phrase (VP) has already been suggested and the noun phrase (NP) is being sought. The determiner (DT) and the noun (N) are then added to the structure in turn to build up the noun phrase. However, a bottom-up approach would follow the sequence shown in Figure 6.2, where the noun phrase is built up before the verb phrase is hypothesised. Here “ball” has been received before “the” has been attached to the output structure.

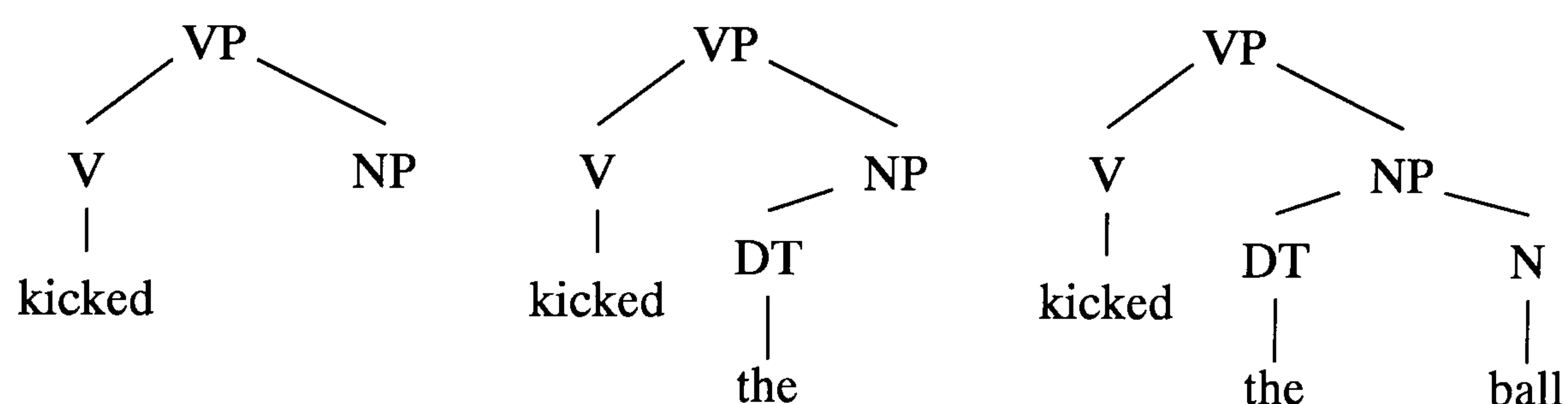


Figure 6.1: An example showing the incrementality of top-down parsing

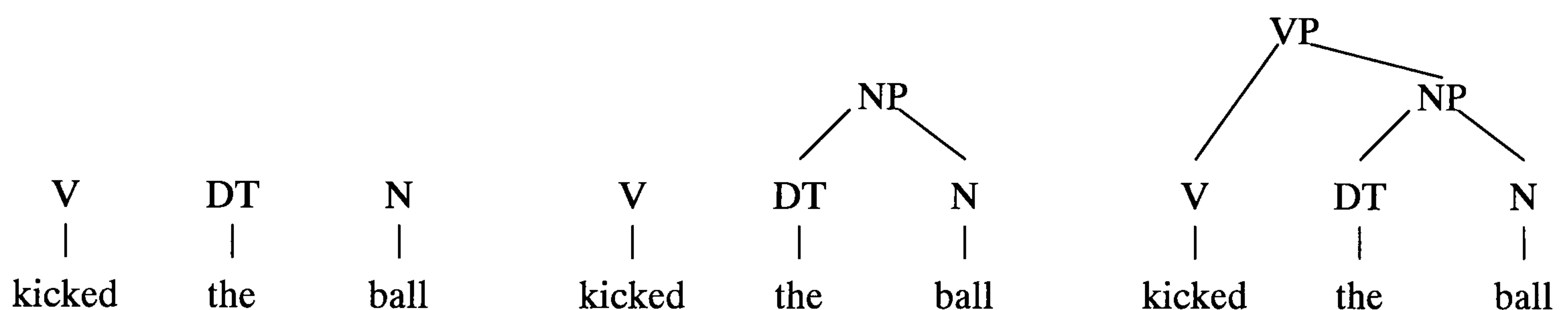


Figure 6.2: An example showing the lack of incrementality of bottom-up parsing

In a learning context, rather than a purely parsing context, it is perhaps unreasonable to insist upon an entirely top-down algorithm. The current state of the grammar does not really allow the top-down prediction required and the constraints on both the parsing and the learning are data-driven, i.e. both processes are currently controlled by what is known about the words. Hence, it is probably not possible to use a purely top-down algorithm. However, the algorithm should at least deal with the input incrementally and be able to be extended to allow the sort of top-down structure prediction shown in the example. Some bottom-up algorithms that meet this less stringent approach to incrementality are discussed below.

However, with respect to efficiency/processing limitations, standard top-down algorithms perform less well. Given a grammar containing left recursion, e.g. some equivalent of the phrase structure rule:

$$NP \rightarrow NP PP$$

then a top-down parser can apply this rule infinitely many times without covering any of



the example. Given that the parser should maintain incrementality, it is not possible to look ahead to the remainder of the sentence to determine how many times it should be applied. This problem is an extreme instance of a general problem with top-down parsers: they can hypothesise a large amount of structure that is in no way relevant to the examples being considered. This makes them, potentially, very computationally inefficient (non-terminating in the case of infinite derivations) and so they are also psychologically implausible.

Bottom-up parsers do not suffer from this problem, as the structure that is built is driven by the data available. Hence, there is no risk of infinite parsing.

It would appear that, from the computational perspective, a bottom-parser with the potential for the addition of some top-down prediction would be the best approach. As this decision is based partly upon the lack of efficiency of a purely top-down parser, then the same reasoning holds with respect to psychological plausibility. The bottom-up parser can be designed to be certain to terminate and, as will be seen below, can be made efficient. Hence a bottom-up parser should be used.

### 6.3.3 Efficient Parsing

In the quest for more efficient parsing algorithms, the most common approach has been to use tabular or chart constructions [57, 4]. The aim is to store information that has already been computed, i.e. syntactic substructures, in data structures so that they can be recalled rather than recalculated. It is also common to store the parse information so that complete parses are not stored, but they can be recovered from this data structure.

The use of these data structures has produced a number of efficient algorithms for parsing natural language, such as the CKY algorithm and the Earley algorithm (see Aho and Ullman [3] for descriptions of these algorithms), both of which parse in cubic time. These parsers are efficient and are generally considered to be psychologically plausible [46, 119]. Crocker [46] indicates that some versions of the chart parsing algorithms which use left-corner tables (i.e. a table of the left-most children which can be derived from each category) can be used to improve efficiency and to show some of the processing limitations of the human sentence-processing system.

The use of a chart also allows each word and its initial input to be attached as it is dealt with (although further structure using the word may be added later). Hence, chart parsers allow a fairly incremental approach with the output even if a bottom-up chart parser is used. It is also possible to add top-down prediction to a bottom-up chart parser if desired [57, 21].

### 6.3.4 The Basic Parsing Algorithm

The discussion above leads to an algorithm that must be left-to-right, bottom-up (although with the potential to have some top-down prediction) and a chart-based parser. The most common algorithms that fit this description are the CKY algorithm and the Earley algorithm [3].

With respect to Categorical Grammars, the chart parsers would appear to need to be bottom up. Currently, the CKY algorithm is easier to modify than the Earley algorithm, as it matches the binary-branching nature of CG and does not require the development of the

equivalent of active charts (charts which include incomplete entries where part of a grammar rule has been applied) and left-corner tables for CGs that are defined for PSGs. Hence, at the moment CLL uses a CKY parser. However, in future, it may be worth investigating some of the improvements that may be achieved using left-corner tables and top-down prediction, which can easily be built into the Earley algorithm.

## 6.4 Approaches to Selecting Most-Likely Parses

The CKY parsing algorithm allows all possible parses to be built for a sentence. In many cases this would be the desired result, but for CLL the set of “best” parses is required. One early approach to finding the “best” parse was determinism, which is discussed below. However, more recently, stochastic methods have been applied in parsing with great success. These are discussed in some detail below. Again Crocker [46] provides some useful analysis with respect to psychological plausibility.

### 6.4.1 Determinism

An early contribution to this field by Marcus [86] was based on the intuition that due to the speed of the human sentence processor and the seeming lack of reanalysis, parsers should be built that do not need to do any reanalysis, i.e. that every parsing decision that is made is unchangeable. If the parser fails, then it should be on sentences which humans fail to process, e.g. garden path sentences. Marcus built a left-to-right, bottom-up parser with three-item look-ahead, using heuristics to determine how to make the parsing decisions.

There are a number of criticisms that can be levelled at such an approach. Firstly, as a bottom-up parser it was not entirely incremental. Secondly, the extent of look-ahead seems implausible and again indicates the lack of incrementality. Thirdly, it relied upon subcategorisation information for decision making, which in verb-final languages would lead to a lot of look-ahead being required. Finally, the black and white distinction of the parse succeeding or failing is not plausible. Experiments suggest that it is much more of a matter of degree. The human sentence processor appears to allow a certain amount of reanalysis.

Despite these fairly strong criticisms, this work has led to the much more plausible notion of *structural monotonicity*, e.g. D-theory, [87] where trees are described in terms of dominance rather than immediate dominance. Structural monotonicity ensures parsing strategies where relations between nodes in the tree are added but never removed (hence monotonicity). Such strategies ensure less reanalysis, giving an approach closer to that of humans. However, the approach is still seeking a single parse. This is not suitable for most settings of CLL, where the aim is to build the  $n$ -best options and choose between them. Once most of the grammatical information is learned, then it would be ideal if the parser settled on only one parse in the way that adult humans do mostly. However, in the process of learning, it seems wise to allow more than one possible syntactic analysis.

In future, it may be worth investigating the possibility of incremental parsing algorithms using Categorical Grammars, which have similarities with D-theory in that the parse is built incrementally. Two general approaches can be considered. The first is to maintain standard



CG categories for words, but extend the set of combination rules to allow incremental interpretation. Steedman's Combinatory Categorical Grammar (CCG) [118, 119], allows this approach. The second approach is to maintain the standard CG rules and modify the categories so that the standard CG rules combine them incrementally, for example the parser of Milward [90]. Given the current state of CLL, both of these options are computationally impractical. Parsers of the type presented by Milward [90] are exponential in the length of the input, which would make CLL too inefficient to be in any way practical. However, extending the set of combination rules would vastly increase the number of possible edge combinations that need to be considered, again making CLL too inefficient to be practical.

### 6.4.2 Stochastic Methods

In Chapter 2, the use of stochastic models of language was motivated. In that chapter, a stochastic version of Categorical Grammar was developed.

The stochastic language model that has been developed allows a stochastic parser to be used to select the  $n$ -best parses generated by the CKY parser using the simple Maximum Likelihood (ML) model, as discussed in Chapter 3. Not only does the model allow ranking of the parses, but it also allows significant reductions to be made in the amount of processing performed by the parser. Collins [44] presents a probabilistic version of the CKY algorithm for selecting the best parse, which ensures that many entries do not need to be added to the data structure (essentially this is the Viterbi algorithm [33]). Here, only the best edge is added to the chart for each particular category. In CLL this approach has been modified to allow the  $n$ -best edges to be added.

So the use of stochastic models of language provides both computational efficiency and a way of ranking analyses. This is exactly what is needed in CLL and hence, a stochastic version of the CKY algorithm is used that returns the  $n$ -best parses.

## 6.5 Conclusions

In conclusion then, the probabilistic CKY parser appears to be a good choice given the requirements of a parser for CLL. It is computationally efficient, especially with the probabilistic constraints. It is also well suited to the binary branching CG formalism. Finally, it is a reasonable choice with respect to psychological plausibility, maintaining a somewhat incremental approach (although the current algorithm should be improved) and using the frequency information available.

There are a number of issues that should be investigated in future. The current algorithm in CLL could be made more incremental fairly easily and thus be rather more psychologically plausible. It would also be interesting to use a left-corner strategy both to improve efficiency and psychological plausibility. There are also a number of possibilities for improving the stochastic model to improve disambiguation by including more linguistic information (see for example the parsers of Bod [18] and Collins [44]). It would also be interesting to investigate more restrictive approaches that tend towards determinism, as these could potentially provide useful constraints that could be used to improve efficiency.

However, the current system has now been described in full and so it remains to describe how it has been tested. Before this, in the next chapter, the psychological context of syntax learning is discussed with respect to CLL.



## Chapter 7

# Human Language Learning

Practical problems in NLP commonly, at least to some degree, intersect with the problems faced by humans with respect to language processing. This should not surprise us, as NLP systems are built to interact in a human-like way with a human artefact – language. It is a valuable and interesting task, therefore, to evaluate the work in this thesis with respect to the psychological evidence available for human language learning [138].

Evaluation of a learning model and system in this way can provide two useful results. Firstly, such evaluation can provide an accurate picture of how close a system is to dealing with human-like problems in human-like ways, i.e. how psychologically plausible a system is. Secondly, such evaluation may suggest possible improvements to the system, either by using psychological information to better inform the design of a system, or to suggest areas where a system could be made more psychologically plausible.

While the work presented in this thesis does, to a large extent, focus on solving a computationally interesting and useful problem, it is considered that it will be informative to compare CLL against the psychological evidence to determine how plausible it is and what possible extensions and modifications could be made to improve CLL.

Therefore, in this chapter, CLL is discussed in the context of the learning model upon which it is based. Each part of the model is briefly discussed with respect to the psychological evidence and suggestions of possible modifications are made.

### 7.1 The Examples

The examples that CLL is provided with are unannotated strings of words that form examples of (presumed) correct English (i.e. the examples are all positive). Such a set of examples is, in many ways, psychologically plausible.

Firstly, it is clear that children hear positive examples, i.e. the adult speech they hear around them, whether directed to them or not. Similarly, it is evident that children are not supplied with negative examples, as this would require a syntactically incorrect statement to be made and in some sense annotated so that the child is aware of its incorrectness, while this may very occasionally occur (someone making a mistake and correcting it), more commonly this is not the case.

However, the issue of mistakes and human spoken language performance does raise the

first issue with respect to the lack of psychological plausibility within the CLL system. The type of language supplied in the examples from the Penn Treebank and from the generated corpora does not really correspond to the type of language available to a child learning language. Most obviously, children do not learn from written, printed, edited, newspaper text, but from broken, noisy, spoken utterances, with mistakes, partial sentences, missing references etc. As such, restricting the system to strings of typed words (and thus removing the entire speech-recognition task), even if there is noise in the corpus, as there is in the Penn Treebank, does not really model the human environment.

There is another issue with respect to the type of examples and the ordering of the examples. With the Penn Treebank, the type of examples used exhibits broad syntactic complexity (although not movement). However, the examples are ordered, so that the shortest examples are processed first. In the psycholinguistic community, there are two common possibilities present with respect to the language complexity of the examples used with children [32, 62, 116]. The first suggests that parents use a simplified version of their language when talking to their children, called either *child directed speech* or *motherese*. Motherese is thought to provide the child with a set of structured language lessons. One could see the use of ordered corpora as a rather rough way of capturing this idea of dealing with simple language first.

The second view stands in contrast to this. Pinker [98] suggests that motherese remains syntactically complex and that there are societies where motherese is not used, but children are still able to learn language. Atkinson [7] also notes that if motherese is simpler then it could be harder for the child to learn, as there is less information available to the child. So the second view would suggest that children are able to learn from normal adult-to-adult spoken language, although they may still do some selection from what they hear [100]. In this case, the corpora supplied to CLL should not be ordered.

Secondly, there is the somewhat complicated issue of annotation of the text. It is clear that there is no explicit annotation of the text with structural information, but there may be two implicit ways in which the text is annotated. The first is intonation, particularly perhaps in the way adults talk to children. Pinker [98] considers this, but determines that it is unclear how the intonation relates to the structure. Secondly, it may be the case that circumstances, i.e. the world state, at the time of utterances may give some semantic annotation. For example, an adult saying “see how we shut the door”, while actually shutting a door gives some semantic annotation [100]. This can be seen as a corollary with the theory that the structure of language models the structure of the world (a theory Pinker puts forward [99]). It is also closely related to the position that children learn language with significant semantic background knowledge (see Pinker again [100] and see the discussion below). Various computational systems have followed this approach (see, for example [113, 114, 115, 132]), where some kind of semantic annotation, especially annotation related to the state of the world at the time of the utterance, is used.

So it may be the case that some sort of annotation could be considered to be available to a child learning language. Alternatively, this annotation could be considered to be semantic background knowledge (see below). However, there is some evidence to suggest that neither of these views are correct. Children who have been severely retarded and have very little



semantic understanding are still able to learn the full range of syntactic structures. Pinker [99] cites a large number of such cases. If one accepts this evidence, then it would seem to suggest that learning syntax does not require complex semantic knowledge to be available.

In conclusion then, while it is true that an unsupervised, positive-only setting is psychologically possible (although there may be some disagreement with respect to annotation), the type of examples provided to CLL, while meeting these requirements, do fail to be plausible in other ways.

## 7.2 The Background Knowledge

The background knowledge supplied to CLL is broad in its nature. There are the sets of CG rules and categories (including complex categories) and, in some cases, the initial bootstrapping lexicons of closed-class words. Such background knowledge has some psychological plausibility.

There are various possibilities with respect to the kind of knowledge humans have when learning language. To some extent, this depends on a person's particular viewpoint. The dichotomy with respect to background knowledge can be summarised by explaining the *nativist* and *empiricist* viewpoints to learning. (See Langley [79] for a more detailed summary.)

The (idealised) nativist viewpoint is that the human learner has only very weak learning capabilities, but comes to the task with a great deal of innate knowledge, thus constraining the number of possibilities that could be learned in a way that allows the weak learning mechanisms to work.

The (idealised) empiricist viewpoint is that the human learner has little or no knowledge when approaching a problem, but has very powerful learning capabilities, such that all the necessary generalisation can take place using the data in the environment.

These are clearly idealised viewpoints and most research in language learning does not hold rigidly to either viewpoint. It should be noted that, for the syntax learning problem, there are possibly two sorts of background knowledge available. The first, about which there is probably the most discussion, is innate knowledge, i.e. knowledge that humans are born with. The second is the knowledge that has already been learnt, e.g. children have already developed a large vocabulary and possibly some syntactic or semantic relations (see Carroll [32], Harley [62], Pinker [100] and Gleitman [59]).

The evidence seems to suggest that there needs to be at least a certain amount of innate background knowledge. Chomsky [37] and Pinker [98] suggest what has become known as the *poverty-of-stimulus argument* [62], which is that the lack of language stimuli available to guide the learning process indicates that there must be innate syntactic principles within the learner. Carroll [32] summarises Pinker's argument [99] in the following way:

1. Positive evidence alone is consistent with too many competing grammars.
2. Negative evidence, which could constrain the problem space, is not generally available.
3. Therefore, some constraints must be innate.

This leads us to the *Subset Problem* described by Culicover [47]. Consider a learner (human or machine) that selects a hypothesis for the language from the evidence supplied. The



first hypothesis may be generalised by a learner on presentation of more positive examples, thus making the original hypothesis cover a subset of the data of the current hypothesis. However, if the learner over-generalises at some stage of the learning process then a problem occurs. All the positive examples will satisfy the over-generalisation (or cause further generalisation) and the hypothesis would cover some negative examples that it should not. In this case, negative examples would be necessary to restrict the hypothesis, as they could highlight the over-generalisation by being deemed positive by the hypothesis, but being known to be negative. However, the setting is such that there are no negative examples. It would seem some innate knowledge is required to prevent this over-generalisation.

Bickerton [16], having analysed creoles, the languages that develop in communities where different nationalities with different languages work alongside each other, developed the *Language Bioprogram Hypothesis* (LBH), which again hypothesises the need for innate language constraints. Initially, in such contexts, a pigeon develops, which is a very limited language that combines elements of both languages found in the community. The pigeon has very limited syntactic structures. The next generation develops the pigeon into a full language – the creole. Bickerton [16] found that the creoles, developing from syntactically impoverished language examples as they do, actually contain syntactic structures not available to the learners from their pigeon environment, but that show a strong similarity to the syntactic structures of other natural languages. Bickerton [16] states:

“the most cogent explanation of this similarity is that it derives from the structure of a species-specific program for language, genetically coded and expressed, in ways still largely mysterious, in the structures and modes of operation of the human brain.”

As well as this theoretical and empirical evidence, there is the fact that there are strong similarities between human languages, which leads to the idea of there being some sort of universal structure controlling language. Chomsky [37] suggests that there is such a thing as a *universal grammar* that resides from birth in the human mind and dictates the types of language that are learned. This has led, over time, to the development of Principles and Parameters theory [47], which is discussed in Chapter 2. There have been a variety of approaches to computational language learning that have used this theory, e.g. [58, 41], which are discussed in Chapter 4.

Alternatively (although the two options are not necessarily mutually exclusive), there is the possibility that, instead of having a large amount of innate syntactic knowledge to inform the learning process, there is instead a set of semantic knowledge. This idea is proposed by Pinker [100], who suggests that children have a semantic model, which language is strongly related to. The semantic model is then used to inform the learning process.

So what aspects of the knowledge supplied to CLL can be psychologically well-motivated? Potentially, a certain amount of CG information could be supplied, on the basis of it being innate syntactic knowledge, i.e. some sort of CG Universal Grammar. However, too much is undoubtedly supplied given the language specific nature of the set of categories. Villavicencio [132] provides a more rigorous model of what kind of syntactic principles can be assumed to be innate using a unification version of CG. This work uses the sort of semantic



supervision described in the previous section in a learning system that sets the parameters of the principles in the model of the Universal Grammar. In future, CLL could be made more psychologically plausible by using a better motivated set of syntactic principles for innate knowledge.

Similarly, to some extent, the initial lexicon can be motivated by the fact that some lexical knowledge should already have been learned [32, 62] and also by the idea that there is a syntactic and a semantic context in which words occur and so some simple relationships may have been built (e.g. [100, 115, 59]). The initial lexicons can be seen as simple syntactic/semantic (both as CG encapsulates both, although the ordering information in CG categories is specific to the syntactic information) bootstrapping lexicons. However, these motivations are not really strong enough for supplying closed-class word lexicons, as there is no reason to assume that initial words should be closed-class (in fact there should probably be a spread of word types although nominals are the most commonly learned [32, 29]). It may also be unreasonable to provide a mapping between words and their CG category, as this may provide more relational information than is appropriate, although this may simply be a way of communicating the syntactic and semantic bootstrapping approaches of Gleitman [59] and Pinker [100] respectively. Hence, CLL could be made more psychologically plausible by using a simpler lexicon, with less knowledge built into it.

In general, although some psychological motivation has been taken into account in determining the kind of background knowledge supplied in the CLL system, the actual knowledge supplied is too great and probably too specific. However, it may be possible in future to gradually modify the background knowledge to be more psychologically plausible.

### 7.3 Learning Mechanisms

CLL essentially uses three learning mechanisms. Firstly, symbolic syntactic constraints are applied by the parser to current and previous examples. Secondly, stochastic methods are used for assigning likelihood values to the parses. Thirdly, compression is used to select a single parse from the set of likely parses.

Significant work has been done with respect to psychologically plausible syntactic processing (see [46] for a summary). This has led to the suggestion that parsers should be incremental both in the way that they receive their information (i.e. they should be left-to-right parsers) and in the way that they build their parses (i.e. the parse should be built in such a way that all new information is attached to the output tree) [46]. It is suggested too, that parsers need to consider processing constraints, i.e. parsers should have similar time and memory constraints as humans and thus show some similar behaviour in parsing difficult sentences [46]. Perhaps one final issue needs to be raised with respect to parsing in a child learning language. This is whether or not the child has a complete parser available at this stage and whether or not complete syntactic processing is yet possible.

The parsing algorithms are evaluated in more detail in Chapter 6, however, from a psychological perspective it would seem that a left-to-right, left-corner chart parser with some top-down lookahead is the best possibility for psychologically plausible parsing [46]. Such parsers are incremental with respect to the input and largely incremental with respect



to the output. They are efficient with respect to time and, according to Crocker [46], exhibit some of the same sort of parsing behaviour as humans with respect to processing limitations.

The probabilistic CKY parser that is used in CLL meets some of these criteria. It is efficient and treats the input incrementally. The output is not treated incrementally yet, but small modifications could allow it to be mostly incremental. There is, as yet, no top-down prediction though and no use of left-corners (or the equivalent for CGs), so modifying the algorithm in both these ways would improve the psychological plausibility of the system.

It should be noted that there has been work on parsing incrementally with CGs. On the one hand there is the extension of CG, Combinatory Categorical Grammar (CCG) [118, 119], where further combination rules are added, which can allow incremental processing. There is also the alternative of not adding rules, but building up one consistent partial structure (with attached semantics), such as the approach of Milward [90]. However, both these extensions would currently causes problems. Using CCG, as has been discussed, is the logical next step to provide better grammatical coverage. However, the increase in the number of rules causes an increase in parsing cost, which the current system could not afford. Similarly, Milward's approach also leads to a less efficient parser, which would be too costly to use with CLL. However, in future, these possibilities could be investigated further.

There is a further issue with respect to parsing that has an impact on the psychological plausibility. Currently, the system stores examples so that reparsing can occur to correct mistakes made earlier in the learning process. This is not plausible due to the storage of examples [14], as children do not show this behaviour – in fact, neither do adults. In future, a system that did not use reparsing should be developed to improve psychological plausibility. However, as has already been mentioned, this approach could be considered to be an approximation to the system receiving large amounts of data.

There are also two statistics-based mechanisms for controlling the learning process: compression and maximum likelihood. A natural objection to the use of statistical models for language is that humans do not seem to need or to be able to perform the necessary word calculations and probability analyses to create models of this kind. Rather it appears that they look at context and what they have previously learned. This is, however, contradicted by some research into the way humans learn and analyse language. As a simple example (given originally by Charniak [33]), if a person is presented with the partial sentence “Jack went to the ...” and the set of words “hospital”, “pink”, “number” and “if”, then one would expect that the person would be able to comment reasonably accurately on the different likelihoods of these words being the next in the sentence, which at least indicates the ability to make qualitative decisions.

Although the previous example is a little crude, Bod [18] claims that frequency is shown to be integral by the psychological studies that show that humans register the frequency with which something happens in language [63, 97], that we prefer to use previous analyses of languages [68], and finally that we prefer more frequently occurring analyses [54]. In other words, we use the frequency of previously occurring information to construct a model for analysing new utterances of language. Bod[18] summarises these studies in the following way.

“a comprehender tends to perceive the most probable analysis of a new input on



the basis of frequencies of previously perceived analyses.”

While it is clear that stochastic models can be used to model these empirical results, it should be recognised that a stochastic model is not the only solution. Clark [42] for example takes the view that the learning of language is “profoundly statistical”, however, the model of language he uses is not a stochastic model. He suggests that when the number of examples of a syntactic concept exceed a certain threshold (the *threshold frequency*) then that concept is accepted as correct. This learning model uses statistical methods, but the model of language does not satisfy the definition of a stochastic model given above, as probabilities are not assigned to any string in the language. Such a model of language and its acquisition accords with the results of the psychological studies cited above, without leading to the conclusion which Bod reached.

Perhaps more strongly, one could suggest that there is theoretically no need for any statistical element at all in language learning and modelling. The psychological studies and Clark’s model show that more evidence helps in the selection of a particular hypothesis as is the case in entirely symbolic learning systems. Evidence at different stages causes one to refine the hypothesis with respect to a particular concept.

It seems clear that the evidence is such that one can use it to present arguments to support a number of approaches. While none of these approaches can be discounted on the psychological evidence, it would be wrong to assume that because an approach is consistent with the evidence it is the correct approach. To quote Magerman[84], one of the proponents of statistical natural language processing, with respect to the use of stochastic models in speech recognition:

“There is no theoretical reason why a rule-based system could not be designed to solve the problem; but no system ever approached the level of coverage needed for general large-vocabulary speaker independent speech recognition.”

This seems to be entirely applicable to the grammar-writing problem in natural language syntax. Perhaps the stronger reason for the use of stochastic models is that it is a good engineering way of handling the problems arising from the difficulty of producing accurate and complete rule bases. However, there seems to be sufficient evidence to suggest that some simple statistical/frequency models are psychologically well motivated. Given the simple state of our current models, it would appear that they are plausible.

One further possibility with respect to learning mechanisms is that, instead of negative *examples*, negative *evidence* is actually available to the child in the form of feedback or correction from parents. Originally, this was based on the Behaviourist viewpoint that everything was acquired by reinforcement and conditioning [62].

This model was tested by Brown and Hanlon [30] by studying transcripts of parent-child conversations. They studied adults’ responses to childrens’ grammatical and ungrammatical sentences and could find no correlation between a child’s grammatical sentences and a parent’s encouragement. In fact the dependency was on whether the utterance was true or not. They even found that parents do not understand children’s well-formed questions better than their ill-formed questions. Pinker [98] reports that these results have been replicated. This can only lead to the conclusion that there is no significant negative evidence available

to the infant attempting to learn syntax. However, these studies appear to be somewhat limited and there have been a number of attempts, from a computational perspective, to allow correction, e.g. Adriaans [2] uses Angluin's Query Learning [6] model to build a syntax learning system. However, the problem with this from an engineering perspective is that providing an oracle of some sort to provide the appropriate correction is very difficult.

It would appear, therefore, that the use of statistical methods within CLL is psychologically justifiable and that the lack of negative evidence is also appropriate (although the Query Learning approach may also be an interesting approach). The main problems with respect to the psychological plausibility of the learning mechanism are related to the symbolic constraints of the parsing and reparsing. The reparsing, in particular, is implausible and should be replaced. It will be seen that such a suggestion is made with respect to the efficiency of the system as well. Once again, the psychological and practical issues would appear to coincide.

## 7.4 Summary

There are, therefore, some good reasons for saying that the approach taken with CLL is psychologically *informed* without perhaps saying that it is psychologically *plausible*. The system is unsupervised, positive-only, incremental to some extent and the background knowledge has some basis in psychological reality. However, as has been suggested, it would be necessary to make some reasonably significant changes to this system to improve its plausibility. In the future some of these changes may be pursued.



# Chapter 8

## The Corpora

Any syntax-learning system will need to be tested with a variety of corpora as it is developed. The approach I have followed is to use a series of corpora of gradually increasing complexity, i.e. where the underlying grammars gradually increase in their complexity. This allowed CLL to be extended appropriately as the corpus complexity increased and highlighted issues (particularly with respect to efficiency and syntactic knowledge) that needed to be addressed within the learning system. In this chapter, the corpora and the processes for building them are described.

There are two types of corpora:

- generated corpora,
- naturally occurring corpora.

Generated corpora are those that have an explicit underlying grammar, from which a set of legal examples are generated to form the corpora. I have used three of this of type corpus, which are described in Section 8.1. Naturally occurring corpora have no explicit grammar from which they are generated. Instead they contain examples of language that have occurred in some “natural” setting, i.e. some human usage like speech or written text. I use the Penn Treebank II [88, 85, 110, 17], from which I extract a number of sub-corpora, which are described in Section 8.2. In Section 8.3, I summarise why the set of corpora described are useful for testing CLL and I discuss some of the options with respect to using other corpora.

### 8.1 Generated Corpora

Of the generated corpora that have been used with the learning system, two have been generated specially for the development of the system and one, the LLL corpus [74], is a generated corpus that has been made available to me<sup>1</sup> for natural language learning experiments.

#### 8.1.1 Simple Generated Corpora

The first corpus (GC1) was built from a context-free phrase-structure grammar (CF-PSG), using a simple random generation algorithm. The CF-PSG (shown in Figure 8.1) covers a

---

<sup>1</sup>Thanks to Dimitar Kazakov for this corpus.

range of simple declarative sentences with intransitive, transitive and ditransitive verbs and with adjectives. The lexicon of the CF-PSG contained 39 words (a few of which are shown in Figure 8.1), including an example of noun-verb ambiguity. The corpus consisted of 500 such sentences with a maximum length of 9 words and an average length of 4.14 words (Figure 8.2 shows examples).

$S \rightarrow NP VP$	$VP \rightarrow Vbar$
$Vbar \rightarrow IV$	$Vbar \rightarrow TV NP$
$Vbar \rightarrow DV NP NP$	$NP \rightarrow PN$
$NP \rightarrow Nbar$	$Nbar \rightarrow Det N$
$N \rightarrow Adj N$	
$PN \rightarrow john$	$Det \rightarrow the$
$N \rightarrow boy$	$Adj \rightarrow small$
$IV \rightarrow ran$	$TV \rightarrow timed$
$DV \rightarrow gave$	

Figure 8.1: The CF-PSG used to generate GC1 with example lexical entries

```
ex([mary, ran]).
ex([john, gave, john, a, boy]).
ex([a, dog, called, the, fish, a, small, ugly, desk]).
```

Figure 8.2: Examples from GC1

This corpus provides an ideal starting point for the learner. The underlying grammar is reasonably simple and should be easily learnt even from the small set of 500 examples.

GC1 allows the testing of the validity of the ideas behind CLL, as the the experiments show (see Chapter 9). However, the underlying grammar is quite simple and for the next increment in complexity, it was decided to generate a new corpus using a more complicated CF-PSG.

The second corpus (GC2) was generated in the same way, but using extra rules to include prepositions (see Figure 8.3), thus making the fragment of English more complicated. The lexicon used for generating the corpus was larger – 44 words in total. Again, 500 examples were generated, this time with a maximum length of 32 words and an average length of 8.72 words (see Figure 8.4 for examples).

$NP \rightarrow Nbar PP$	$VP \rightarrow Vbar PP$
$PP \rightarrow P NP$	
$P \rightarrow on$	

Figure 8.3: The extra rules required for generating GC2 with example lexical entries

```
ex([the, fish, with, a, elephant, gave, banks, a, dog, with, a, bigger, statue]).
ex([a, elephant, with, jim, walked, on, a, desk]).
ex([the, girl, kissed, the, computer, on, a, fish]).
```

Figure 8.4: Examples from GC2



This corpus can be used to investigate how the learner performs with a more complex underlying grammar. Moreover, issues with respect to the learning of attachments can be investigated using the prepositional rules and lexical entries.

### 8.1.2 The LLL Corpus

Having initially used corpora generated from hand-written CF-PSGs, it would have been possible to continue to extend the underlying grammar to include both a larger lexicon and more syntactic phenomena. However, this seemed to have potentially been completed already with the LLL corpus [74]. This is a corpus of generated sentences for a substantial fragment of English. It is annotated with a certain amount of semantic information, which was ignored for the purposes of our experiments. The corpus contains 554 sentences, including a lot of movement (e.g. topicalised and question sentences). Examples are shown in Figure 8.5. The corpus is perhaps somewhat small – especially as sentences with movement were not covered by the learner. However, it provided a useful next step in the process of developing and testing the learner.

```
ex([which, rough, reports, above, hilary, wrote, hilary, in, sandy, beside, which,
telephone])).
ex([inside, no, secretary, wont, all, small, machines, stop])).
ex([which, old, report, disappears])).
```

Figure 8.5: Examples from the LLL corpus

The corpus was used both in its entirety (LLL(M)) and also for experiments without movement examples (LLL), which led to a significantly smaller corpus. Chapter 9 contains the details of how the LLL and the LLL(M) corpora were used.

## 8.2 Naturally Occurring Corpora

The generated corpora allow a set of initial experiments with CLL. However, they can only be considered as a starting point, for if the system is intended to learn the syntax for natural language, then it is necessary to test how the system performs on naturally occurring language. So the next set of corpora used to test CLL were derived from a corpus of naturally occurring language – the Penn Treebank (described in Section 8.2.1). However, to continue the step-wise increase in complexity, these corpora do not contain the full complexity of natural language. Instead, only a restricted set of examples – restricted by length and to some extent by syntactic content – were extracted from the corpus and translated into an appropriate form for the learner. The extraction and translation process is described in Section 8.2.2. It is a modular process and allows various corpora to be built. Those that have been built for use in the experiments are described in Section 8.2.3.

### 8.2.1 The Penn Treebank

The Penn Treebank [88, 85, 17] is a corpus of over 4.5 million words of American English, which have been annotated with both parts-of-speech and syntactic trees (a list of the parts-of-speech is provided in Appendix A for reference). The majority of the corpus is taken from

Tag	Functional Label
Text Categories	
HLN	headlines and datelines
LST	list markers
TTL	titles
Grammatical Functions	
CLF	true clefts
NOM	non NPs that function as NPs
ADV	clausal and NP adverbials
LGS	logical subjects in passives
PRD	non VP predicates
SBJ	surface subject
TPC	topicalised and fronted constituents
CLR	closely related
Semantic Roles	
VOC	vocatives
DIR	direction and trajectory
LOC	location
MNR	manner
PRP	purpose and reason
TMP	temporal phrases

Table 8.1: The functional labels in the Penn Treebank II

the Wall Street Journal, so it is, in fact, representative of written English and perhaps more specifically, of written American-English from a newspaper. While this is a fairly specific type of natural language, the Penn Treebank is, for a number of reasons, a good corpus upon which to perform experiments. Firstly, it provides a wide range of syntactic structures for a system to learn. Secondly, the corpus is marked with appropriate syntactic information to allow comparisons with learned structure and labels (although see Chapter 10 for a discussion of the potential complexities involved in this). Thirdly the corpus is large, providing a large amount and variety of data from which to learn. Finally, probably for the preceding reasons, the Penn Treebank has become the most commonly used corpus for testing syntax-based NLP systems (e.g. part-of-speech taggers and parsers), which means that results achieved using the Penn Treebank will be easier to compare with the results of alternative systems.

The Penn Treebank II is being used [17, 85]. This version attempts to address the problem of predicate argument structure, which previous versions had ignored. Figure 8.6 shows an example from the Penn Treebank with both syntactic structure and part-of-speech annotation included. While the documentation is clear that predicate argument structure is not explicitly marked [85], the annotation includes a set of functional labels that relate to the role of constituents in the sentence. The list of the functional labels is reproduced (from Marcus *et al* [85]) in Table 8.1. These labels are attached to the standard constituent label after a “-”, as can be seen from the example in Figure 8.6, e.g. NP-SBJ and PP-LOC. The labels, while not a complete annotation of predicate argument structure, do give a strong indication. Collins [44] and Xia [145] have both used heuristics, based on the annotation, to build predicate-argument structure.



```

( (S
  (NP-SBJ-1
    (NP (NNS Yields) )
    (PP (IN on)
      (NP (JJ money-market) (JJ mutual) (NNS funds) )))
  (VP (VBD continued)
    (S
      (NP-SBJ (-NONE- *-1) )
      (VP (TO to)
        (VP (VB slide) )))
    (, ,)
    (PP-LOC (IN amid)
      (NP (NNS signs)
        (SBAR (IN that)
          (S
            (NP-SBJ (NN portfolio) (NNS managers) )
            (VP (VBP expect)
              (NP
                (NP (JJ further) (NNS declines) )
                (PP-LOC (IN in)
                  (NP (NN interest) (NNS rates) ))))))))
          (. .) ))
  )

```

Figure 8.6: An example from the Penn Treebank with structural and part-of-speech annotation

The other important issue with respect to the annotation of the treebank is the marking of null elements and the co-indexing of the traces. Each null element in a tree is marked where the word should be with either the label *\** or the label *\*T\** (for cases of topicalisation and WH-movement). These labels are also given an index, i.e. a number attached to them after a “-”, e.g. *\*-1*. This number relates to the index on the category of the word that has moved, e.g. NP-SBJ-1. Figure 8.6 includes an example of a null element with a co-indexed trace. The explicit annotation of sentences with null elements is useful in the corpus extraction procedure, as these sentences are eliminated from the extracted corpora.

Perhaps the one weakness of the Penn Treebank is that users find that the annotation is inconsistent and incorrect at many points (Marcus *et al* [88, 85] comment on the inconsistency). This means that the corpus, while exceedingly useful, needs to be considered to have noise in the annotation. The noise will be relevant when it comes to the evaluation of the learning system (Chapter 10).

In summary, the Penn Treebank provides a large and well-annotated corpus that has become a standard with which to test systems. There are some limitations: the type of text available and also the noise to be found in the annotation. However, despite these limitations it is clearly a very appropriate corpus to use for testing CLL and provides a good source from which corpora can be extracted.

### 8.2.2 The Extraction Procedure

The next step is to outline the process I have used to extract corpora from the Penn Treebank. There are three output corpora that will be useful:

- the sentence corpus,
- the part-of-speech corpus,
- the tree corpus.

The contents of these corpora are fairly self-evident. They each contain the same sentence examples in the same order, selected from the Penn Treebank under the restrictions set by the extraction process. The sentence corpus is a file of the examples with no annotation. The part-of-speech corpus, is a file of the examples where each word is annotated with the (possibly modified) Penn Treebank part-of-speech tag. Finally, the tree corpus is a file of the examples extracted with the (modified) tree annotation of the Penn Treebank (including the part-of-speech annotation). The modifications to the annotation of both the part-of-speech and the tree corpora relate to the learning process and are described below.

The extraction of these corpora follows a sequential and modular approach, which is shown in Figure 8.7. The input to the system is the set of Penn Treebank files (with both the part-of-speech and the tree annotation), which are to be used for building the corpus, and the maximum length (in words) of the examples to be included in the output corpora. Each of the modules can be included or not when extracting a corpus, which allows the user to obtain the corpus of their choice. Each module is described in turn below, showing the effect each stage has on the example shown in Figure 8.8.

**Translation** The translation process mostly just converts the bracketing from a Lisp format to a Prolog format. This involves converting the parentheses (“(” and “)”) into square brackets (“[” and “]”) and adding commas between arguments. Apostrophes are added around elements of the annotation with special meaning in Prolog, e.g. words starting with capital letters, which would be treated as variables otherwise. This ensures that such items are treated as atoms. Also, all apostrophes in the Penn Treebank are translated to hyphens, as the apostrophe is a special character in Prolog. Similarly, commas that occur within numbers in the Treebank are removed, as they have a special meaning in Prolog. Figure 8.9 shows the example after it has passed through the translation stage.

**Movement Removal** CLL is not currently supplied with an appropriate grammar for dealing properly with movement. All examples with movement are clearly marked in the Penn Treebank, so it is a fairly simple matter to remove them. All examples containing the part-of-speech `-NONE-`, which indicates that the example has a moved element, are removed. The example in Figure 8.9 does not contain any such part-of-speech and so it is retained.

**Fragment Removal** The Penn Treebank marks fragments, i.e. incomplete sentences and phrases, with the `FRAG` label. Currently the grammar and parser cannot deal with these incomplete utterances and so all sentences containing the `FRAG` label are removed. As the example in Figure 8.9 does not contain the `FRAG` label, it is retained.

**Possessive Combination** Possessives in the Penn Treebank are annotated as two parts, the root word marked with its category and then the possessive ending (now either “-s” or



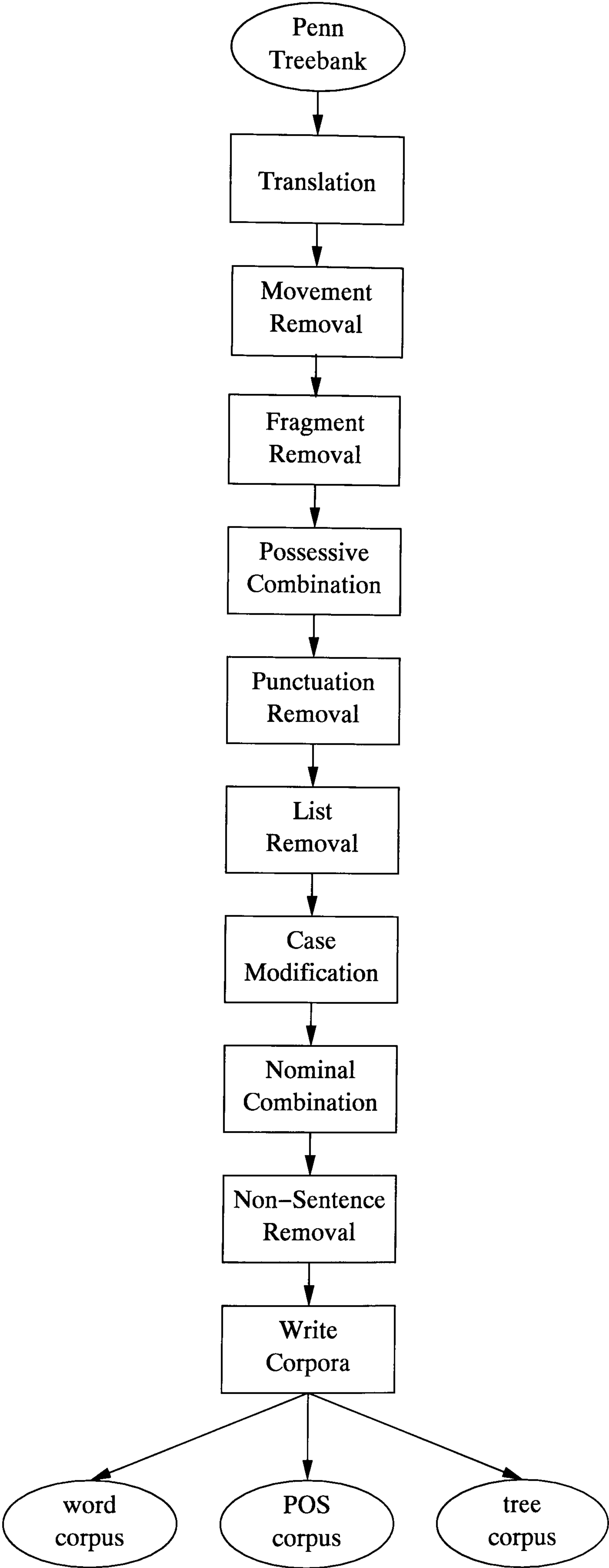


Figure 8.7: The structure of the system for extracting corpora from the Penn Treebank

```
( (S (NP-SBJ (NNP W.R.) (NNP Grace) ) (VP (VBZ holds) (NP (NP (CD
  three) ) (PP (IN of) (NP (NP (NNP Grace) (NNP Energy) (POS 's) )
  (CD seven) (NN board) (NNS seats) )))) (. .) ))
```

Figure 8.8: Another example from the Penn Treebank with structural and part-of-speech marking

```
[[S, [NP-SBJ, [NNP, 'W.R.'], [NNP, 'Grace']], [VP, [VBZ, holds],
  [NP, [NP, [CD, three]], [PP, [IN, of], [NP,
  [NP, [NNP, 'Grace'], [NNP, 'Energy'], [POS, -s]],
  [CD, seven], [NN, board], [NNS, seats]]]]], [., .]]]
```

Figure 8.9: The example after the translation module is applied

“s-” as the apostrophes have been translated) marked with the POS part-of-speech. These two parts are joined into one word with one category – POSX – where X is the category of the root word. The parts-of-speech are combined because the CG that is used with CLL treats possessives as one word. In the example, there is the possessive NNP “Energy” which is joined with its possessive ending to give “Energy-s” and is given the category POSNNP. The result of possessive combination on the example is shown in Figure 8.10.

**Punctuation Removal** Punctuation is currently ignored by the learning system (although it could possibly be used to indicate phrasing), so a module to remove all punctuation is included. The module simply uses a list of all the punctuation parts-of-speech (see Appendix A) and removes any occurrences of them. Figure 8.11 shows the results of this module when applied to the example. In this case, the removal of the full-stop at the end of the sentence is the only change.

**List Removal** In some circumstances the Penn Treebank uses the LST label to indicate a list of items. These are commonly involved in a somewhat complex co-ordination structure, which the current CG would not handle very elegantly. Hence, examples containing LST components are removed from the corpus. As the example does not contain the LST label it is not removed.

**Case Modification** The case modification module is used to replace the capital letter at the start of words that are not proper nouns. This reduces the number of “different” words, e.g. “the” and “The”, which are held in the corpus. In the example, all words starting with capitals are proper nouns and so there are no changes. It should be noted that these modifications will have an impact upon the size of the lexicons learned.

**Nominal Combination** The learning system is designed to avoid the complexities of compound nominals, as is the annotation of the treebank on the whole. Hence, there is a

```
[[S, [NP-SBJ, [NNP, 'W.R.'], [NNP, 'Grace']], [VP, [VBZ, holds],
  [NP, [NP, [CD, three]], [PP, [IN, of], [NP,
  [NP, [NNP, 'Grace'], [POSNNP, 'Energy-s']],
  [CD, seven], [NN, board], [NNS, seats]]]]], [., .]]]
```

Figure 8.10: The example with possessives combined



```
[[S,
  [NP-SBJ, [NNP, 'W.R.'], [NNP, 'Grace']],
  [VP, [VBZ, holds],
    [NP, [NP, [CD, three]],
      [PP, [IN, of],
        [NP,
          [NP, [NNP, 'Grace'], [POSNNP, 'Energy-s']],
          [CD, seven], [NN, board], [NNS, seats]]]]]]
]]
```

Figure 8.11: The example with punctuation removed

```
[[S,
  [NP-SBJ, [NC, 'W.R._Grace']],
  [VP, [VBZ, holds],
    [NP, [NP, [CD, three]],
      [PP, [IN, of],
        [NP, [NP, [NNP, 'Grace'], [POSNNP, 'Energy-s']],
          [NC, seven_board_seats]]]]]]
]]
```

Figure 8.12: The example with nominals joined

module which joins adjacent nominals in the same subtree, i.e. directly under the same NP root, and gives them the label NC. The module does some pattern matching over subtrees to determine if there are suitably adjacent nominals. The nominals are joined using the ‘\_’ character between the words. It should be noted that not all adjacent nominals are joined, only those in the same NP-rooted subtree. Figure 8.12 shows the effects of nominal combination on the example. Two compounds are formed, “W.R.\_Grace” and “seven\_board\_seats”. The example also shows that possessives are not currently included in this combination process. The module could easily be extended to include possessives.

**Non-Sentence Removal** Currently, CLL only deals with examples that are sentences. In particular, independent phrases that are not sentences, some of which exist within the Penn Treebank, are not to be used as examples. A module is included to exclude these examples by removing all examples with a root that is not labelled with a category beginning with S, e.g. S and SBAR, which are the Penn Treebank labels to indicate sentences. As the example is a sentence, it is not removed.

**Writing Corpora** Each of the corpora is written to a file at this stage. For the sentence and part-of-speech corpora the structural annotation is stripped away, leaving simply a list of words and word-part-of-speech pairs respectively. For the sentence corpora, the list of words is written with the option of marking the nouns (any part-of-speech beginning with N) and verbs (any part-of-speech beginning with V) in the example with a simple N and V marker respectively. The two output options for the example are given in Figure 8.13. The list of word-part-of-speech pairs is written to the file for the part-of-speech corpus. This corpus can be used for generating a set of examples that are part-of-speech sequences rather than word sequences, although experiments have not yet been performed using these examples.

```
ex(['W.R._Grace', holds, three, of, 'Grace', 'Energy-s', seven_board_seats])
ex([(N, 'W.R._Grace'), (V, holds), three, of, (N, 'Grace'), 'Energy-s', (N,
seven_board_seats)])
```

Figure 8.13: The two possible versions of the example written to the sentence corpora

It is at this stage of the process, once all possible transformations have occurred, that the length constraint is applied. Only sentences of a length less than or equal to the user specified maximum (15 and 25 for the corpora used in the experiments) are written to the corpora.

Chapter 9 discusses the use of these corpora. The tree annotation that was the output of the last module is written to the tree corpus.

**Post-Processing** After the corpora are generated a certain amount of post-processing is generally performed. An appropriate number of examples are selected (see Section 8.2.3) and are then ordered on length. This places the examples that require the least processing early, in an attempt to improve the efficiency of CLL. While it is not necessary to do this and may be considered psychologically implausible (although there is no reason to assume that a child does not select simple examples or parts of examples to learn from first), it has the advantage of speeding up the experiments. Experiments where this ordering has not taken place should also be completed in the future.

### 8.2.3 The Extracted Corpora

The corpora extracted from the Penn Treebank essentially vary along three dimensions:

- maximum example size,
- noun/verb annotation,
- number of examples.

The majority of experiments were carried out on corpora with a maximum sentence length of 15 words. This is long enough to capture the majority of syntactic constructions available, but short enough to allow the experiments to run quickly, so that various settings of the parameters of CLL could be investigated. A small set of experiments was performed on a corpus where the maximum sentence length was set to 25 to investigate the best approaches on a corpus containing larger examples.

Secondly, experiments were carried out both with and without the N and V annotation for nouns and verbs respectively. So two sets of corpora were required.

Finally, the number of examples varied. Corpora of 5,000 examples were used as input to CLL, whereas corpora of 1,000 examples were used to test the lexicons learned.

A set of experiments have also been performed using a somewhat noisy corpus extracted from the Penn Treebank. This was achieved using an earlier version of the extraction software. Sentence fragments are included in the corpus, the nominal joining module performs less joins and not all punctuation is removed. This allows for experiments to be performed



Name	Content	No. of Examples	Max. Length	Ave. Length	Brackets	Annotation	Noise
PC1(tr)	text	5000	15	8.74	25.71	none	✓
PC1(ts)	text	1000	15	8.7	25.60	none	✓
PC2(tr)	text	5000	15	9.56	28.03	none	×
PC2(ts)	text	1000	15	9.47	27.74	none	×
PC3(tr)	text	5000	15	9.56	28.03	NV	×
PC3(ts)	text	1000	15	9.47	27.74	NV	×
PC4(tr)	text	5000	25	12.54	36.59	NV	×
PC4(ts)	text	1000	25	11.93	34.88	NV	×

Table 8.2: A summary of the naturally occurring corpora used in the experiments

to determine the effect of noise on the learning system, which should give some idea of the robustness.

The corpora extracted from the Penn Treebank allowed the investigation of a variety of parameters in the CLL system on naturally occurring language. The details of the experiments in which they were used are given in Chapter 9. The corpora are summarised in Table 8.2 and they are named to allow them to be referred to easily. The “tr” on the names indicates the training set and the “ts” indicates the test set. Example lengths are measured in number of words. Brackets is the average number of pairs of brackets per example.

### 8.3 Discussion

In this chapter I have outlined the corpora used in the various experiments described in Chapter 9. The aim has been to provide a series of corpora of increasing syntactic complexity and lexicon size, so as to test and develop CLL. A feature of the series was the use of two different types of corpora. Firstly, corpora generated from hand-built grammars were used, which allows testing where the underlying grammar is known. Secondly, corpora of naturally occurring examples, extracted from the Penn Treebank were used. These have the advantage of containing the syntax found in natural language, which the system aims to learn. The Penn Treebank has the further advantages of being appropriately annotated for comparison and also of being a standard corpus with which most systems are tested.

One issue with all of these corpora is that they are relatively small (all are less than 100,000 words). Currently this is for practical reasons. The current size makes it possible to complete experiments in a reasonable time (approximately a week with PC1, PC2 and PC3 and around two weeks with PC4) with the current version of CLL. This risks compromising the effectiveness of the learner, as the data will be relatively sparse and in fact has an impact on the results, as it is difficult for CLL to perform the required generalisation on sparse data. Two alternatives can be investigated in the future. The simplest would be to have corpora which were sequences of parts-of-speech tags, which is a common way of reducing the sparseness of data employed, for example, by both Osborne and Briscoe [96] and Bangalore and Joshi [10], but unfortunately it also reduces the lexical ambiguity rather significantly. Preliminary investigations along these lines indicate that the level of reparsing

becomes much greater (due to the fact that examples have many more “words” in common). Hence, it will be necessary to consider restricting the reparsing in some way. A second approach would be to attempt to increase the efficiency of the parser so that CLL could deal with more examples, including those of greater length. One approach to this would be to consider using less complete parsing techniques (i.e. heuristically restricting the search for parses) or using partial parsing techniques [1].

While a wide variety of corpora have been discussed and used in the experiments, there are a number of clear possibilities for alternatives. Most obviously, it would be interesting to perform experiments on the full Penn Treebank, including examples with null elements, examples of all sizes and examples that are not sentences. This is not a trivial task and the amount of processing involved would make experiments too computationally expensive with the current system.

However, perhaps more interesting from a psychological point of view would be to use corpora of spoken utterances rather than written. This would more closely match the environment in which a child learns. Again this would be difficult to implement with the current system, as it would involve dealing with incomplete utterances and utterances that form only parts of sentences. However, this would be an interesting further development of CLL.

It might also be useful to perform experiments upon sequences of corpora with increasing lexicon size and syntactic complexity, using the lexicons from earlier experiments as initial lexicons for later experiments. In effect, this might be something like the sequence of structured language lessons suggested by the motherese hypothesis (see Chapter 7). It may, after all, be the case that children have some technique for selecting simple utterances or parts of utterances first.

However, despite the large number of further possibilities for alternative corpora which could be used for experiments, those that are presented here provide a wide-ranging set which can be used to perform instructive experiments and to show the validity of the learning system that has been developed.



## Chapter 9

# The Experiments

In this chapter, I describe the experiments that have been performed using various different settings of CLL. As one might imagine, there are a huge number of types of experiment that could be performed, so it is important to determine exactly what experiments have been performed and why.

In Section 9.1, the parameters which are varied in the experiments are described and motivated. The aim has been to perform experiments with a fairly broad range of parameter settings to give a good indication of the performance of CLL in various contexts. Following this, the progression of the experiments is described in Sections 9.2 and 9.3, from the early experiments with a simple setting on simple corpora, to the large scale experiments on corpora extracted from the Penn Treebank. There are many further experiments that could be performed, so, in Section 9.4, I suggest some experiments, which I see as important to further investigate the model and the system. Finally, in Section 9.5, some conclusions on the experiments are discussed.

### 9.1 Experimental Parameters

#### 9.1.1 Initial Lexicon

Some mention has already been made of the initial lexicon that may be provided to the learner for bootstrapping purposes. It is clearly likely to be quite important in providing a useful initial bias for CLL. Compared to building either a full lexicon, or annotating a corpus, this is a relatively small task. From investigating the psychological setting of the problem it seemed that children may have some kind of partial lexicon present when they come to learning, and so it seemed reasonable to consider some options for including initial partial lexicons.

In the earlier experiments on the generated corpora, no initial lexicon was used. However, this later proved to be impractical, so two further settings have been investigated. Firstly we considered using a closed-class word lexicon. Closed-class words, such as determiners and prepositions, have categories which have a finite and known number of words, unlike, for example, verbs and nouns.

The closed-class lexicons consisted of determiners, prepositions and conjunctions and the possible categories those words could take. These lexical entries were assigned an approx-

imate probability distribution as they were built, usually by giving a uniform distribution across all the categories for a word. This distribution is currently not adjusted during the learning process (although the lexicon learned can have a different distribution). It would be better in future to either set more informed prior probabilities, or to allow these probabilities to be modified as CLL progresses, or both. However, as an approximation, this seems to give satisfactory results.

Two versions of the closed-class-word lexicon have been used. The first, CCW1 is simply a small lexicon of common closed-class words. The lexicon contains 31 lexical entries, which corresponds to 21 words. The second closed-class lexicon, CCW2, was extracted from the Penn Treebank. The lexicon contains most of the words that are assigned the appropriate part-of-speech category (DT, IN, CC) in the extracted corpora and so, in some senses, can be considered an approximately complete lexicon for these categories. CCW2 is therefore much larger, containing 348 lexical entries for 136 words. It is assumed that this will allow CLL to be more accurate in the assignments it makes right from the start. Appendix B contains these two lexicons.

The second type of “initial” lexicon provides a grouping of words that are nouns and verbs. In fact, this is implemented by annotation of the corpus with the labels N and V for noun and verbs respectively. This does slightly more than just provide us with a group of nouns and verbs, it also indicates occurrences in the corpus, which in turn performs a certain amount of disambiguation as these words could perhaps occur as other parts-of-speech, e.g. “run” could be a nominal in a phrase like “the long run”, but a verb in “I run a long way”. The parser uses these annotations as described below to limit the ambiguity in the learning process. One might consider this to be a level of supervision, but it is significantly weaker than most of the approaches used in alternative systems, as was seen in Chapter 4.

### 9.1.2 Parser

Various implementations of the parser have been used, from a simple and not particularly efficient probabilistic chart parser in the earliest reported work [134, 135] to the efficient  $n$ -best probabilistic CKY based implementation of the later work (PCKY). The differences in these parsers make little difference to the grammars learnt (although earlier versions were not guaranteed to find the highest probability categories). They do, however, make a huge difference to the speed with which the results are achieved. In fact, the progression of parsers was followed to allow larger scale experiments to be performed. The parser has already been described in detail in Chapter 6.

However, the parser has three parameters which can be investigated:

- beam size,
- initial lexicon use,
- smoothing approaches.

The beam size, i.e. the maximum number of the most probable parses produced for a sentence, can be set to any positive integer. The larger the beam, the more options with which the learner is presented for possible new lexicons. However, while more options are



likely to allow greater compression, it is also the case that the higher the number of options, the larger the number of improbable options presented. The learner does not currently use their probability and so may use improbable parses to inaccurately compress. Hence, it would seem to be necessary to perform some investigation of the appropriate beam size.

Currently, experiments have been performed with beam sizes 1, 2 and 4, which seems adequate to indicate the trend, although further experimentation with higher beam values may be useful.

Secondly, on all experiments apart from the earliest experiments on the GC1, the GC2 and the LLL corpora, CLL is provided either with an initial bootstrapping lexicon of closed-class words, or the corpus is annotated to indicate where there are nouns and verbs in the sentences, or both these resources.

With the closed-class-word initial lexicons there are two possible areas for variation. Firstly, with the words in the initial lexicon, they may or may not be allowed to be assigned categories other than those assigned in the lexicon. Secondly, when unknown words occur in examples, the system may or may not assign closed-class categories to them. Both of these issues concern the same trade-off. The advantage of allowing words to be assigned a larger variety of categories is that more options can be investigated and evaluated. There are two disadvantages. Firstly, investigating more options means more time needs to be spent on each parse. As the parsing is so critical to the efficiency of the learner, this may not be wise. Secondly, relaxing the assignment constraints reduces the learning bias and may allow the learner to investigate mostly implausible options. Currently, to investigate the use of the initial lexicons, two settings have been applied to the learner. The first is that the initial lexicon is used such that the words in it can use only the categories assigned to them in that lexicon and no other words can take the closed-class categories. In the second, the initial lexicon is used as a guide. Words in it can still only take the categories assigned in the lexicon, but other words may take closed-class-word categories. In future a less restrictive approach still could be used, where the words in the initial lexicon are allowed to take other categories as well.

When the initial lexicon is effectively only a nominal and verbal grouping, then there is only one possible setting of the parser, which is that, on the occurrence of an N or V, the parser is restricted to the CG categories available for nominals and verbs respectively.

Finally, there is the issue of smoothing. As has already been described (see Chapter 5), a very simple version of smoothing is used to allow unknown word-category pairs to be considered by the parser. This is obviously vital in an essentially unsupervised algorithm, which starts with either no lexicon, or a minimal lexicon. No alternatives have yet been investigated for smoothing, but it is an area which ought to be studied in future.

### 9.1.3 Compression Metric

The compression metric is clearly important, because it is a major driving force between selecting extensions to a learned lexicon (along with the generation of the most probable parses). The metric is essentially an heuristic, because it is a matter of intuition as to exactly what should be compressed – there are an infinite number of possible metrics which could be used, but those used in CLL were selected for their intuitive value.

One fundamental decision to be made with respect to the compression metric, is what is to be compressed. By this, I mean, are we aiming to learn the smallest possible lexicon (which will be denoted lexicon compression), or the smallest lexicon representing the corpus (which will be denoted corpus compression). The two metrics used have already been discussed in detail in Chapters 3 and 5.

It should be noted that the two metrics are fairly simple, which has the advantage that they can be calculated very efficiently. However, there may many other features that one may want to include. For example, it may be desirable to include some notion of category size if one wishes to bias the learner toward learning smaller categories (although very early experiments indicated this would prevent the learning of ditransitive verbs). It may also be desirable to include information from the probabilistic parser on the likelihood of the parse being considered. However, the two metrics that have been used thus far are intuitive, effective and efficient, which makes them good candidates.

#### 9.1.4 CG Category Databases

The learner is provided with a set of lexical categories, i.e. the categories that can be assigned to words. In the initial experiments below, a small set of categories (CATDB1) was used for efficiency. This contains 12 categories, although only those that were applicable to the corpus in question were used. This early set was hand-built to match the early CF-PSGs that were used to generate corpora.

Following this, the set was extended significantly (by hand), so that it could be expected to deal with the greater demands of the corpora extracted from the Penn Treebank. This database of categories (CATDB2) contains 30 categories.

Finally, a set of categories (CATDB3) was extracted from the CG annotated corpus produced by the formalism translation process described in Chapter 10. All the categories in the lexicon were extracted, but only the 45 most frequently used categories were included in CATDB3.

It should be noted that the greater the number of categories, the broader coverage the grammar will have, but also the less constrained the learning system will be. CATDB3 contains a large number of categories that only occur infrequently, the likelihood is that using so many categories will lead to the parsing being under-constrained and so lots of spurious parses. The spurious parses could have a negative effect on the lexicons learned by adding spurious lexical entries.

All of the category databases described in this section are reproduced in Appendix C.

#### 9.1.5 Noun-Phrase Handling

Compound noun-phrases are not easily handled with any grammar and this is the case with CG also [144]. The current corpora make some attempt to pre-combine noun phrases prior to processing, as this is simply a difficult problem to handle. However, not all compound noun-phrases are combined.

To some extent then, compound noun phrases need to be handled by the grammar. Two approaches suggest themselves. Firstly, compound noun phrases usually have an internal



structure of a head and modifiers/complements. The grammar could simply use the set of categories provided for this, in particular  $n/n$  and  $np/np$ . The alternative, is to use a noun phrase joining rule:

$$np\ np \Rightarrow np$$

The former has the advantage of efficiency and simplicity. The latter has the advantage of more accurate coverage and of assigning noun phrases more sensible – nominal – labels. Both options are pursued.

It is possible that a more sensible option would be to attempt to apply the joining rules lazily, i.e. only when it is absolutely necessary. This should be pursued in future, as should the use of another rule for joining nouns, as this could improve the handling of these phrases.

### 9.1.6 Corpora

Chapter 8 contains descriptions of various corpora, all of which are used as input into the system. In some ways, the different corpora are the major difference between experiments and so, they have tended to be grouped with respect to which corpus CLL is being applied to.

## 9.2 Initial Experiments

The earliest experiments (presented in [134, 135, 136]) were all on the generated corpora – GC1, GC2, LLL and LLL(m). These were used mostly as an indicator for what should be attempted next. They used basic settings that were not altered much. The lexicon was compressed, not the corpus, the beam was set to 2 and no noun-phrase handling was necessary, as only simple noun phrases were present in the corpora.

The initial lexicon was varied, as it was either empty, or had a small number of entries when CCW1 was used. The category database available to the learner was CATDB1, although categories were removed if they were not going to be useful for parsing the given corpus. The corpora were obviously varied even using a version of the LLL corpus that included movement (LLL(m)).

## 9.3 Large Scale Experiments

Following the initial experiments, CLL was significantly developed for use on much larger corpora. The whole system was made significantly more efficient, especially the parser. These experiments have all been performed on the corpora extracted from the Penn Treebank.

Essentially, a set of 10 experiments was performed on each of the unannotated corpora PC1 and PC2, that is those extracted from the Penn Treebank with word length 15 or less. These are summarised in Table 9.1. In this table, note that Fixed CCW implies that words cannot be assigned the categories included in the initial lexicon. The aim of this scheme of experiments is to use Experiment 1 as a baseline and vary the parameters around this. Experiments where more than one parameter is changed were included, as they were performed to pursue better results on the basis of the results of the other experiments. It

Number	Initial Lexicon	Fixed CCW	Categories	Beam	Compression	NP rules
1	CCW1	No	CATDB2	2	Lexicon	Yes
2	CCW2	No	CATDB2	2	Lexicon	Yes
3	CCW1	Yes	CATDB2	2	Lexicon	Yes
4	CCW2	Yes	CATDB2	2	Lexicon	Yes
5	CCW1	No	CATDB3	2	Lexicon	Yes
6	CCW1	No	CATDB2	1	Lexicon	Yes
7	CCW1	No	CATDB2	4	Lexicon	Yes
8	CCW1	No	CATDB2	2	Corpus	Yes
9	CCW1	No	CATDB2	4	Corpus	Yes
10	CCW1	No	CATDB2	2	Lexicon	No

Table 9.1: The experiment plan when an initial lexicon is used

Number	Categories	Beam	Compression	NP rules
1	CATDB2	2	Lexicon	Yes
5	CATDB3	2	Lexicon	Yes
6	CATDB2	1	Lexicon	Yes
7	CATDB2	4	Lexicon	Yes
8	CATDB2	2	Corpus	Yes
9	CATDB2	4	Corpus	Yes
10	CATDB2	2	Lexicon	No

Table 9.2: The experiment plan when an initial lexicon is not used

is clear that there are a huge number of options for further experiments (720 experiments could be performed by just varying the settings of the parameters in Table 9.1). It is hoped that those shown will give a good flavour of how each parameter affects the system.

Two sets of experiments were performed with PC3, the corpus with nouns and verbs marked. A set of experiments like that shown in Table 9.1 were performed, so that CLL had both an initial bootstrapping lexicon and the noun and verb marking of the corpus to guide the learning process. Another set of experiments were performed that did not have any initial bootstrapping lexicon. Table 9.2 summarises the set of experiments that did not use an initial lexicon. These are essentially the same type of experiments as those in Table 9.1 but with the initial lexicon and fixed CCW parameters removed, as they are no longer relevant when there is no initial lexicon. As the experiments are similar in the two tables, the numbering has been unified such that experiments in Table 9.2 have the same number as experiments with the same parameter setting (plus an initial lexicon setting of CCW1 and a fixed CCW setting of No) in Table 9.1. This may help somewhat in the comparison.

Finally, experiments 1 and 4 were performed on PC4, the NV-annotated corpus, with examples of word length 25 or less. Initial lexicons were used for these experiments. These experiments take significantly longer to run and are more recent so fewer have been completed as yet. The two experiments should provide a some indication of how CLL performs with the longer examples.



## 9.4 Future Experiments

A very large number of future experiments have been suggested throughout this chapter, either by varying the current parameters in different ways, or by adding new parameters, or new settings for current parameters. It would have been impossible to investigate all of these settings and it is suggested that it may be impractical to do so in future.

Instead, I will suggest the experiments that I consider most worth pursuing. Firstly it would be useful to investigate experiments on the corpora PC4 and the corpora of part-of-speech sequences, as only a few experiments have been carried out thus far (none have been completed with the part-of-speech sequences). It would also be interesting to perform experiments using alternative compression metrics, especially a formalised MDL approach (the corpus compression method is close to this). Ideally a further set of experiments would be on corpora including movement, which would require a different parser and a larger category database. Also, experiments that in some way marry the probabilistic information from the parser and the compression information would be particularly interesting. Perhaps, it would be possible to weight the lexicon change caused by a new parse by the probability of that parse. Finally, further experiments on larger corpora, perhaps bootstrapped with initial lexicons learned from smaller corpora would give a further indication of how well CLL would scale.

## 9.5 Conclusions

Despite it being clear that there are many possible experiments, hopefully it is also clear that the approach followed in the work presented here is both sensible and likely to provide useful results. The experiments should at least provide a strong indication of how the different parameters affect CLL, although it will only provide a small amount of information on how the parameters interact.

Chapter 11 will show that these experiments have been both fairly general, but directed enough to indicate the most effective settings for CLL.

# Chapter 10

## Evaluation Methods

CLL has now been presented and justified and the experiments to be performed have been discussed and settled upon. However, before presenting the results it seems important to discuss in some detail just how they will be evaluated, especially as the evaluation process is by no means trivial. In this chapter, I present the techniques and metrics used upon the output of CLL along with why they are used and what they can show.

Before the evaluation techniques can be discussed, it needs to be made clear exactly what data is produced by the learning system and what form it is in. The data available will, to some extent, determine the evaluation techniques that are applicable. In Section 10.1 the results given by the system and their format are discussed in some detail.

With this information in mind, the evaluation techniques and metrics that will be used are described in Section 10.2, I also discuss the reasons for using them and what exactly they may be used to conclude. This is followed by a discussion, in Section 10.3, of how these metrics are applied to the results.

It becomes clear from the discussion of the types of evaluation to be carried out, that there is a great need for CG-annotated gold-standard versions of the corpora extracted from the Penn Treebank. Section 10.4 presents a new method for achieving this goal, which could easily be generalised for other corpus translation tasks. The work in this section has been taken almost directly from work that has already been published [139].

Finally, in Section 10.5, the evaluation of the experiments is reviewed and some suggestions for further evaluation are presented.

### 10.1 The Results and Their Format

CLL can output a large amount of data as it progresses through a corpus. The idea is to output the entire state of the learner to a file, as this allows it to be restarted if there are any problems that cause undesirable early termination. This means that the lexicon and all the stored parses for each of the examples is available. For convenience, the chosen parse for each of the examples is written to a separate file.

The lexicon is a set of three argument facts where the arguments are the word, the CG category it can take and the frequency with which it has taken it. Figure 10.1 gives some examples of lexical entries that are taken from the output of the learning system. They are



```
lex(sits, bs(s,np), 1).
lex(darkness, np, 1).
lex(include, bs(s,np), 1).
lex(two_ironies, np, 1).
lex(intrude, bs(s,np), 1).
lex(american_express_card_charge_volume, np, 1).
lex('12%', bs(bs(s,np),bs(s,np)), 1).
lex(berry, np, 1).
```

Figure 10.1: Examples of entries in the lexicon from the output of the learning system

```
ex(41, [s,[[np,[[np,wcrs],[np,plans]]],[bs(s,np),'ad-unit_sale']]]).
ex(42, [s,[[np,wellington],[bs(s,np),[[fs(bs(s,np),np),was],
[ np,closed]]]]]).
ex(43, [s,[[np,'computer-generated'],[bs(s,np),[[fs(bs(s,np),np),videos],
[ np,help]]]]]).
ex(44, [s,[[np,'short-term'],[bs(s,np),[[fs(bs(s,np),np),rates],
[ np,increased]]]]]).
```

Figure 10.2: Examples of parses from the output of the learning system

retained in the form that they are in while CLL is being executed, which was described in Chapter 5.

The parses are bracketed CG derivations, where words are assigned a CG category and then are bracketed into the phrase that results from applications of the CG rules. The categories that result from these derivations label these bracketings. Figure 10.2 shows some example parses taken from the output of the learning system. Again these are stored in the format that has been described already in Chapter 5.

The state of the learner, including the current lexicon and parses, can be output after every fifty examples to allow for restarting with only a small amount of work being lost. This, however, also allows us to evaluate the results at different stages in the learning process, which will be interesting with respect to investigating the development of the lexicon.

## 10.2 Evaluation Techniques and Metrics

It has already been noted that the experiments fall naturally into two groups. The earlier experiments were performed on the simpler generated corpora. Given the simpler nature of the the experiments, the evaluation was somewhat simpler. The evaluation of these experiments is discussed in Section 10.2.1.

The results of the larger scale Penn-Treebank experiments also need to be evaluated. Given the size of the experiments, they required more complicated evaluation techniques. With these experiments, the success of the learning system is determined by applying a number of techniques to the results to get values for metrics. In the current work metrics are used that allow the following features to be measured:

1. lexicon size,

2. lexicon ambiguity,
3. lexical accuracy,
4. structural accuracy.

Each of these metrics provide different information which gives a guide as to the quality of the CG lexicon that the system has learned when compared against the values of some gold standard. The metrics, the methods for calculating values for them and the standards against which they can be compared are discussed in turn in Sections 10.2.2–10.2.5.

### 10.2.1 Evaluation of the Early Experiments

With the earlier experiments, where only the generated corpora (GC1, GC2, LLL, LLL(M)) were used [134, 135, 136], the results could be evaluated by hand. Two metrics were used. The first was lexicon accuracy. The lexicon that had been learned was examined and the percentage of linguistically plausible entries was calculated (based upon personal, and therefore to some extent, subjective, judgement). This is a good measure of the quality of the lexicon that was built, but is perhaps somewhat subjective.

The second evaluation metric used is parse accuracy. This is the strict metric of whether the whole parse is plausibly labelled and bracketed given the sentence. Again this metric has a subjective element, but it was very useful in evaluating the early performance of CLL.

These metrics were essentially somewhat *ad-hoc* methods of evaluating the early results presented by CLL. Such metrics were appropriate in the context of having no gold standard available for either labelling or bracketing (although it would probably have been possible to obtain bracketings). They provided a good indication of the effectiveness of the approach and were needed to indicate if it was worth proceeding with the ideas underlying CLL. More formal methods of evaluation were clearly needed. These were used for the Penn Treebank experiments and are described below.

### 10.2.2 Lexicon Size

The simplest measure of lexicon size is the number of lexical entries found in the lexicon. This can clearly be calculated very easily by counting the number of facts in the lexicon contained within the output from the learning system.

One part of the hypothesis being investigated, is that compressing the lexicon is a valid way of constraining a learning system to search a space which includes linguistically useful CG lexicons. The lexicon size is a simple way of showing how small a lexicon is and can easily be compared with the results from other experiments. Firstly, this indicates the need for a gold-standard corpus annotated with CG categories. Section 10.4 presents a method for obtaining such a corpus, but the comparison is still dependent on the category set allowed for each corpus (the translation system allows the creation of new categories) and whether all examples are available for comparison (both the translation system and CLL fail on some of the examples in the corpus).

The lexicon size is simply the number of entries in the lexicon. The immediate value of this metric is that it provides an indication of the compression of the lexicon that has



occurred. The degree of compression can be determined by calculating the size of the lexicon compared to the number of words that have been processed to build it, i.e. the total number of words in the examples processed by the learner, or the sum of all the frequencies maintained in the lexicon. The underlying assumption here is that the worst case lexicon would have a unique entry for each word of each example.

This can also be used in conjunction with the other metrics to perform interesting comparisons with results of previous experiments. For example, how do parse and lexical accuracy vary with lexicon size?

### 10.2.3 Lexicon Ambiguity

Lexicon ambiguity is the average number of categories per word in the lexicon. Calculating this from the CG labelled version of the Penn Treebank provides a measure of the complexity of the problem <sup>1</sup>. Again different experiments can be compared with this gold standard.

### 10.2.4 Lexical Accuracy

The lexical accuracy is the percentage of words that have been assigned the correct CG category (again when compared with the CG-annotated gold-standard corpora).

$$\frac{\text{number of words with the correct categories}}{\text{total number of words}} \times 100$$

This labelling accuracy gives a good idea both of how well the mapping between words and categories has been built by CLL, as well as indicating how well the lexicon can be used for producing well-labelled parses.

It is somewhat awkward to determine a sensible baseline for lexical accuracy. In a system where a complete probabilistic lexicon is provided, the most obvious baseline would be to use the most probable category for each word. However, as this is in no way provided to the learner, it would seem an unfair baseline to set. One possibility would be the expected accuracy of random category assignment based on a uniform distribution over the categories. This is a good baseline given the unsupervised nature of the system and the fact that a complete lexicon is not provided. The system should, of course, do better than this, given that it is restricted by either an initial bootstrapping lexicon, or the noun and verb annotation (or both). An alternative would be to assign the most common category, *np*, to all words, which would set a higher baseline. In some sense, CLL might be considered to be provided with a similar bias, as the *np* category is preferred for new words, so this could be considered to be a more accurate baseline. Both will be given for comparison.

### 10.2.5 Structural Accuracy

There are various methods of measuring parse accuracy (see Goodman's summary [61]). Of these I have selected *consistent brackets* or *crossing brackets*. Using the Penn Treebank parses as the gold standard, this calculates the number of constituents in the learned parse

---

<sup>1</sup>I am grateful to Ted Briscoe for suggesting that some measure of the complexity of the search space would be useful.

which are not disallowed by constituents in the correct tree (i.e. do not have pairs brackets that have one bracket from a constituent in the correct parse within them and the other outside).

It will be interesting to calculate the average number of crosses per sentence, i.e. the average *crossing-bracket rate*. This metric provides us with a notion of how well the categorial lexicon has been built, i.e. how effective is the lexicon for parsing. It also allows comparison with other work on parsing the Penn Treebank, e.g. Collins [44]. It should also be noted that this is a sensible compromise on accuracy, using both a labelling and a structural metric in the evaluation.

The crossing-bracket rates will be compared with the crossing-bracket rates for a set of heuristics for building bracketings for trees. The three heuristics are right-branching, left-branching and random bracketing. All of these are binary-branching heuristics like the parses produced by CLL. Right-branching structures always branch on the right-hand node of each pair of children. Left-branching structures always branch on the left node. The random heuristic selects which node to branch at random. Figure 10.3 gives examples of each.

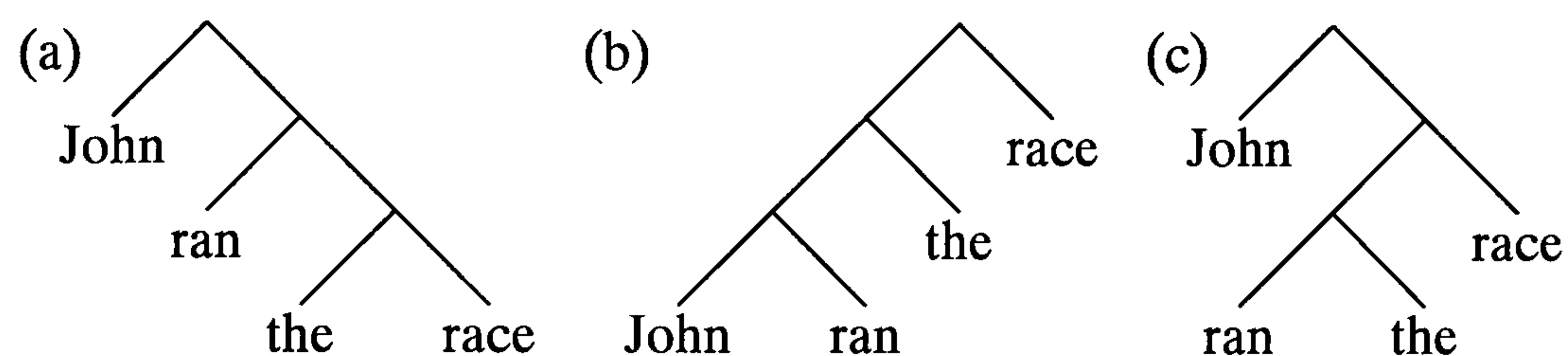


Figure 10.3: Branching examples: (a) right branching, (b) left branching, (c) random branching

English is, essentially, a right-branching language and so this heuristic should be close to the correct structure for the examples. In particular, when compared with the Penn Treebank bracketing, which is fairly flat and removes a certain amount of the non-right-branching structure, the right-branching heuristic should perform well. Both the left-branching heuristic and the random heuristic should perform badly as they are contrary to the structure of English.

CLL is not supplied with any branching heuristic, although there may be some bias in the types of categories supplied in the category databases. However, if CLL performs better than the random heuristic it has clearly learned some correct structural information and if the results with CLL improve upon the right-branching heuristic, then the system is performing well.

It is common to calculate precision (the percentage of brackets hypothesised by the system that are correct) and recall (the percentage of brackets in the gold standard corpus that have been hypothesised by the system) with respect to bracketing. However, the CLL system produces binary-branching trees, which are not really comparable with the trees given by the Penn Treebank. The Penn Treebank trees are much flatter, so CLL will hypothesise many more brackets. Given this, precision and recall measures are unlikely to be very informative, as the structures they are comparing are not intended to be the same (in particular, CLL will be penalised on precision values, as many of its brackets, while potentially entirely correct, will be counted as incorrect because the Penn Treebank does not contain them). The



crossing-bracket measure would, therefore, appear to be a more useful method of comparing the learned parses against those in the Penn Treebank.

### 10.3 Applying the Metrics

The metrics are applied in a simple way. The lexicon size and ambiguity are calculated for the final lexicon extracted in each experiment.

Lexical accuracy and structural accuracy are applied both to the annotated corpus that is the result of the learning process (i.e. the annotated training corpus) and also to a test corpus. The test corpus is a set of examples that have been held back. These are parsed using the parser from CLL and the lexicon that has been learned. The parser returns only the best parse in this case.

### 10.4 Translation of the Penn Treebank Annotation

Annotated corpora have become a vital tool for Natural Language Processing (NLP) systems, as they provide both a standard against which results can be evaluated and a resource from which to extract linguistic information, e.g. lexicons. This is especially true in any NLP task that requires the annotation of examples, e.g. part-of-speech tagging, parsing and semantic annotation, where it is vital to have a correct standard against which to compare the results of systems attempting to solve the task. Similarly, it is crucial in a language learning context, where what is learned can be used to annotate examples, e.g. syntax learning and lexical learning. In this case the learned artifact is used to annotate the examples, which can then be compared against the correctly annotated version. Hence, correctly annotated corpora are vital for the evaluation of a very large number of NLP tasks.

Unfortunately, there are often no suitably annotated corpora for a given task. For example, the Penn Treebank [88, 85, 17] provides a large corpus of syntactically annotated examples mostly from the Wall Street Journal. It is an excellent resource for tasks dealing with the syntax of written English. However, if the annotation formalism (a phrase-structure grammar with some simple features) does not match that of one's NLP system, it is of very little use. For example, suppose a parser using Categorical Grammar [144, 118] is developed and applied to the examples in the corpus. While the bracketing of the examples will bear a strong relationship to the bracketing of the treebank, the labelling of the lexical items and the inner nodes of the tree will be entirely different and no labelling evaluation will be possible.

However, intuitively, plenty of syntactic information is available. In fact, for most evaluation, all the syntactic information required is available, but in the wrong form. It seems obvious that a system for translating the syntactic information between formalisms would be a useful tool.

Here, we present a system that translates the annotation of the Penn Treebank from the standard phrase structure annotation to a Categorical Grammar (CG) annotation and in the process induces large scale CG lexicons. It is a data-driven multi-pass system that uses both predefined rules and machine-learning techniques to translate the trees and in the process



induce a large-scale CG lexicon. The system is designed to produce the lexical annotations for the sentences without null elements (i.e. without movement) from the Penn Treebank, so that these could be used to evaluate the results of CLL.

The system has four major features. Firstly, there is significant control over how the treebank is annotated. This is vital if the results are to be used for evaluation. Secondly, the system prevents propagation of translation errors throughout the trees by being data-driven. Thirdly, the system deals elegantly with erroneous annotation, even providing a degree of self-correction. Finally, the approach is general enough to apply to other similar problems.

The system is compared with a top-down alternative based on the algorithm of Hockenmaier *et al* [67], which is currently the system which has been applied to the most similar task, although it is really for CG lexicon extraction. The comparison suggests that the algorithm presented here gives more compact and linguistically elegant solutions. Investigation also indicates that the corpus produced is effectively translated for its purpose.

In Section 10.4.1, other work in the area is briefly reviewed. In Section 10.4.2, the precise translation task is described. This is followed in Section 10.4.3 with a detailed description of the algorithms used for this task and some discussion as to their appropriateness. The results from the experiments are in Section 10.4.4. Finally, in Section 10.4.5, the results are discussed along with the contributions of the work and some suggestions for future work.

### 10.4.1 Previous Work on Corpus Translation

In this section, the previous work that has been done that is useful with respect to corpus translation is considered. Some of this, as it is essentially supervised grammar learning, is also discussed in Chapter 4.

The most appropriate work to consider within this context is the grammar extraction literature. Perhaps the earliest example is the approach of Charniak [34], who simply extracted a context-free grammar by reading off the production rules implied by the trees in the Penn Treebank. While not translating the formalism of the treebank, this has led to work extracting grammars of different formalisms.

The majority of work is based on the most obvious extension of the Charniak approach, which is to extract subtree-based grammars, e.g. the Data-Oriented Parsing (DOP) approach [18], or extracting Lexicalised Tree Adjoining Grammars (LTAGs), or more generally Lexicalised Tree Grammars (LTGs) [94, 145, 36]. Each approach involves a process that splits up the annotated trees in the treebank into a set of subtrees that define the grammar. These approaches still continue to work with the syntactic data in the same form as it is found in the corpora.

A slightly different approach has been followed by Krotov *et al* [77], where they extract the grammar from the Penn Treebank like Charniak, but then compact it. This provides a smaller grammar of similar quality to a grammar that has not been compacted, when a linguistically motivated compaction is used. However, the formalism remains unchanged. Similarly, Johnson [70] modifies the labelling of the Penn Treebank, but remains within a CFG framework.

Hockenmaier *et al* [67], although to some extent following the approach of Xia [145] where LTAGs are extracted, have pursued an alternative by extracting Combinatory Categorical



Grammar (CCG) [118, 144] lexicons from the Penn Treebank. In this case, the data in the treebank is truly translated into another formalism providing an entire CCG annotation for the corpus based on a top-down algorithm. The lexicon is built by reading off the lexical assignments made for each tree. This is the most closely related work to this research, especially as it translates into a formalism very closely related to CG.

The algorithm presented by Hockenmaier *et al* [67] has been used to build a top-down system against which to compare our data-driven system. The algorithms are both described in detail in Section 10.4.3.

### 10.4.2 The Task

Given a subset of the examples from the Penn Treebank annotated with syntactic and part-of-speech information (slightly modified), the system should return the examples annotated with the correct CG categories attached to the words of the sentence and the lexicons these imply.

The context of the task explains some parts of its definition. The translated corpus is to be used as a standard against which to compare the lexical annotation (i.e. the categories assigned to the words) of the output of an unsupervised CG learner that annotated the words of the examples with CG categories and then extracts a probabilistic lexicon (i.e. the output of CLL). Hence, there is no need for specific tree annotation. The learner currently uses a slightly modified subset of the treebank, which is described below.

#### 10.4.2.1 The Corpus

The systems are applied to examples from the Penn Treebank [88, 85, 17] a corpus of over 4.5 million words of American English annotated with both part-of-speech and syntactic tree information.

To be exact, we are using the Treebank II version [17, 85], which attempts to address the problem of complement/adjunct distinction, which previous versions had ignored. While the documentation is clear that the complement/adjunct structure is not explicitly marked [85], the annotation includes a set of labels that relate to the role of a particular constituent in the sentence. These labels are attached to the standard constituent label and it is possible to use heuristics to determine the probable complement/adjunct structure in the trees [44, 145], which is obviously useful in translating the annotation.

The full Penn Treebank is not being used. As mentioned already, the current research only uses sentences without null elements (i.e. without movement) from the treebank and does not include any of the sentence fragments. However, as Categorical Grammar formalisms do not usually change the lexical entries of words to deal with movement, but use further rules [144, 118, 67], the lexicons learned here will be valid over corpora with movement. The extracted corpus, C1, in fact contains 5000 of the declarative sentences of fifteen words or less (although the sentence length makes little difference to either of the translation procedures described) from the Wall Street Journal section of the treebank. To give an indication of the complexity of the corpus, the number of tokens, i.e. the total number of words including repetitions of the same word, is 47,782. The total number of unique words, i.e. not including

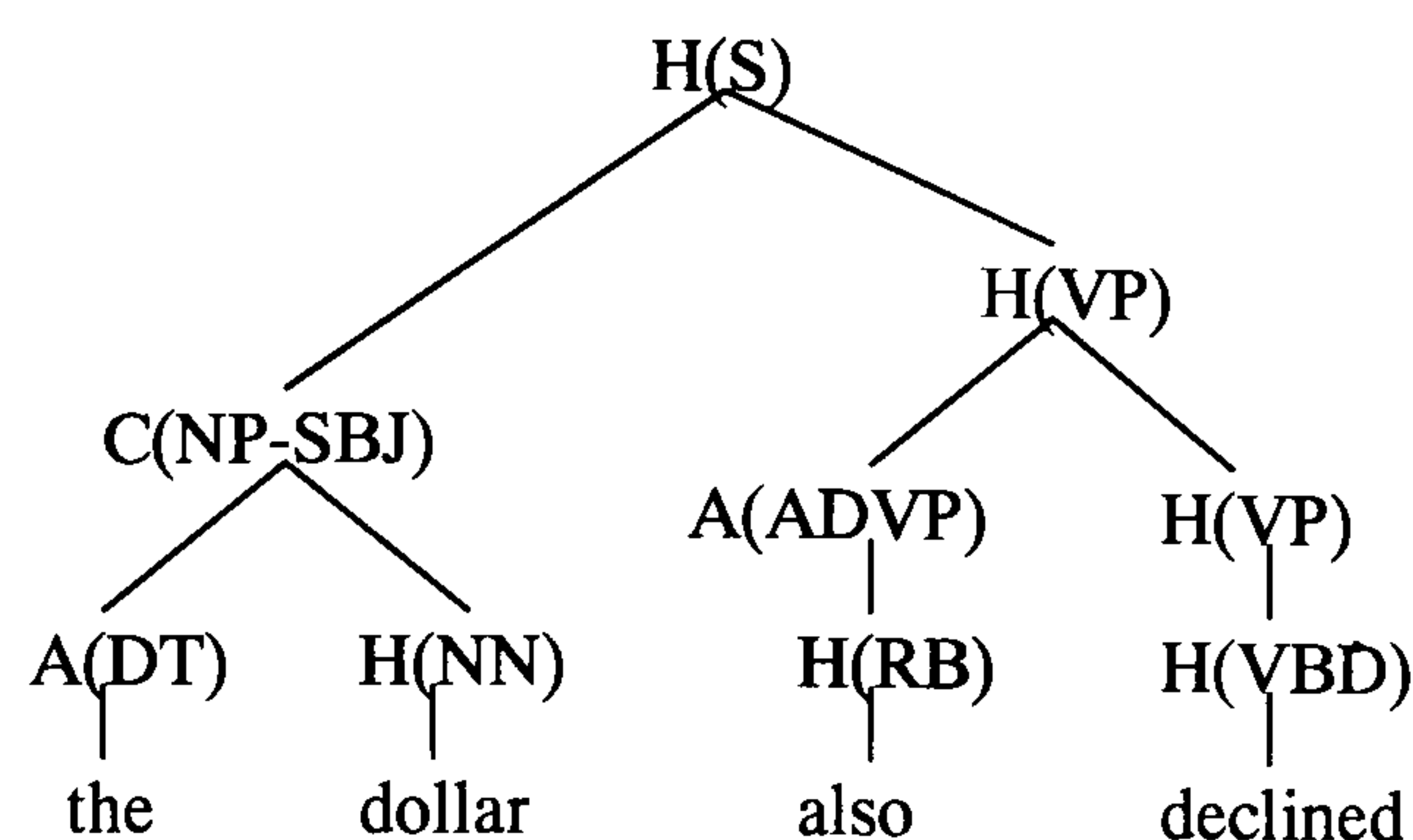


Figure 10.4: A tree with constituents marked

repetitions of the same word, is 12,277. We also extracted C2, a 1000 example corpus (also of declarative sentences from the Wall Street Journal section) with 9467 tokens and 3731 words, which is used in the evaluation process.

The corpora also have some small modifications, which mean that adjacent nominals in the same subtree are combined to form a single nominal and the punctuation is removed (see Chapter 8 for details). These modifications are made for use with CLL [136, 137] to simplify the learning process. They may also slightly simplify the translation process, but it is necessary for the corpus annotation that we want.

### 10.4.3 Alternative Approaches

This section presents the two approaches to translation that are being compared. Firstly, there is the top-down method, which is a version of the algorithm described by Hockenmaier *et al* [67], but used for translating into simple (AB) CG rather than the Steedman's Combinatory Categorical Grammar (CCG) [118]. The algorithm here does not need to deal with movement, as the corpus does not contain any. The atomic *pp* category is included in the CG with this approach, but not with our approach, as it is a convenient shorthand for the prepositional phrase category.

The second approach is a multiple-pass data-driven system. Rules for translating the trees are applied in order of complexity starting with simple part-of-speech translation and finishing with a category generation stage.

#### 10.4.3.1 Top-Down Category Generation

The algorithm has two stages.

**Mark constituents** All the nodes of all trees are marked with their roles, i.e. as heads, complements or adjuncts. While Hockenmaier *et al* [67] are unclear, it is assumed that this is achieved using heuristics. Collins [44] describes such a set of heuristics, which are used with some minor modifications for CG and the changed Penn Treebank annotation. Figure 10.4 shows an example of an annotated tree.

**Assign categories** This is a recursive top-down process, where the top category in the tree is an *s*. The category of the complements is determined by a mapping between Treebank labels and categories, e.g. NP in the treebank becomes *np*. Hockenmaier *et al* [67] do not provide the mapping, so it was built specially for this system. This mapping led to the inclusion of the *pp* category as shorthand for prepositional complements. It should



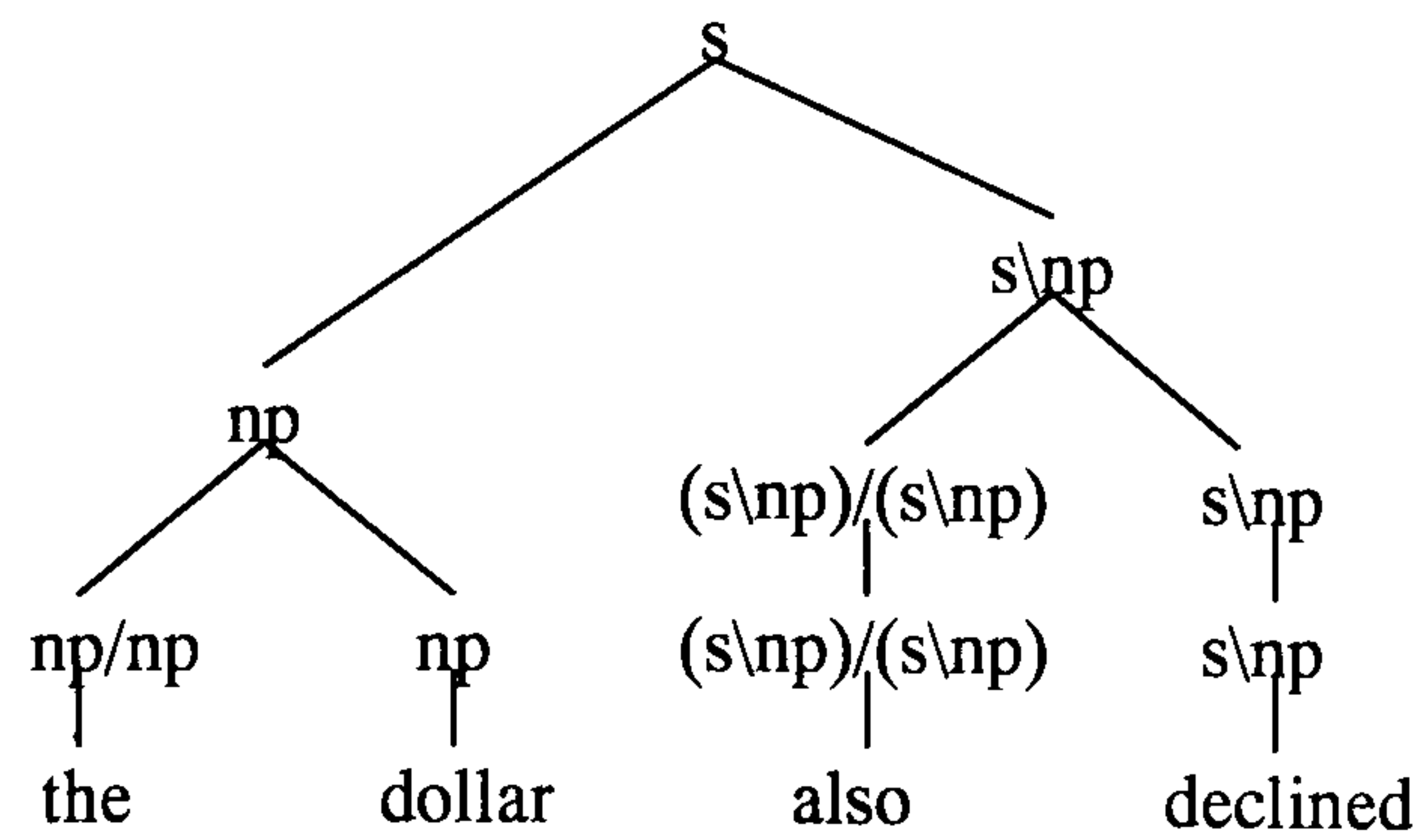


Figure 10.5: An example with categories assigned

make no difference to the annotation process, but could lead to the generation of a few more categories. The head child of a subtree is given the category of the parent plus the complements required, which are found by looking first to the left of the head and then to the right, and adding them in the order they should be processed in. Finally, adjuncts are assigned the generic  $X/X$  or  $X\backslash X$  where  $X$  is the head category with the complements removed which have been dealt with before the adjunct is processed. Figure 10.5 shows an example of a tree with the categories assigned to it.

This algorithm has several advantages. It is simple and robust and has been shown by Hockenmaier *et al* [67] to provide good lexical annotation leading to useful CCG lexicons.

However, it has two main disadvantages. Firstly, there is no control over category generation other than the rather weak constraints of the formalism and the heuristic syntactic roles. This is likely to lead to some linguistically implausible annotation. Secondly, the top-down nature of the algorithm is likely to lead to any translation errors being propagated down the tree, which will lead to some unusual and large categories, as Hockenmaier *et al* [67] report.

#### 10.4.3.2 Bottom-Up Sequential

Our system uses a four stage process, where the type of translation changes at each stage.

**Stage 1: Parts-of-Speech** This is the simplest level of translation. The mapping between the Penn Treebank part-of-speech annotation and the CG category annotation is many-to-many, but some parts-of-speech can be translated directly into categories using simple rules, e.g. the following rule states that words with the determiner part of speech (DT) can be translated into the CG category  $np/n$ .

$$DT \rightarrow np/n$$

The system passes through the full set of examples and translates the appropriate parts-of-speech. See Figure 10.6 for an example of the output of this stage.

**Stage 2: Subtrees** The next pass through the data allows more complex rules to be used. Consider the part-of-speech label NNS, used in the Penn Treebank annotation scheme to indicate a plural noun. Its syntactic role can be that of a simple noun ( $n$ ) or a noun phrase ( $np$ ), so we need a mechanism for choosing between these two possibilities.

The most obvious mechanism is to use the surrounding subtree to provide the context to select the correct rule. If the NNS tag is part of a noun phrase which begins with

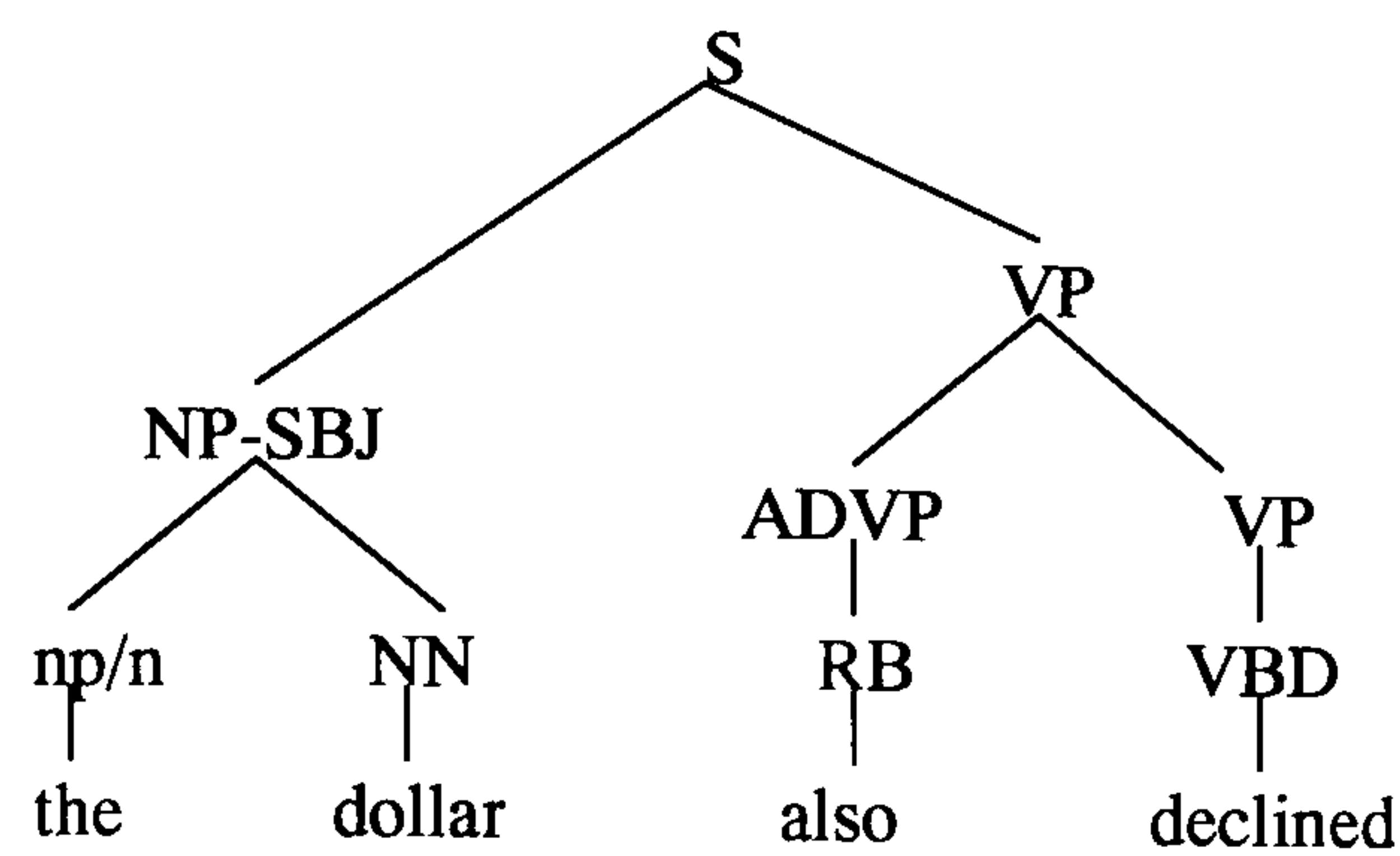


Figure 10.6: Example of the output of Stage 1

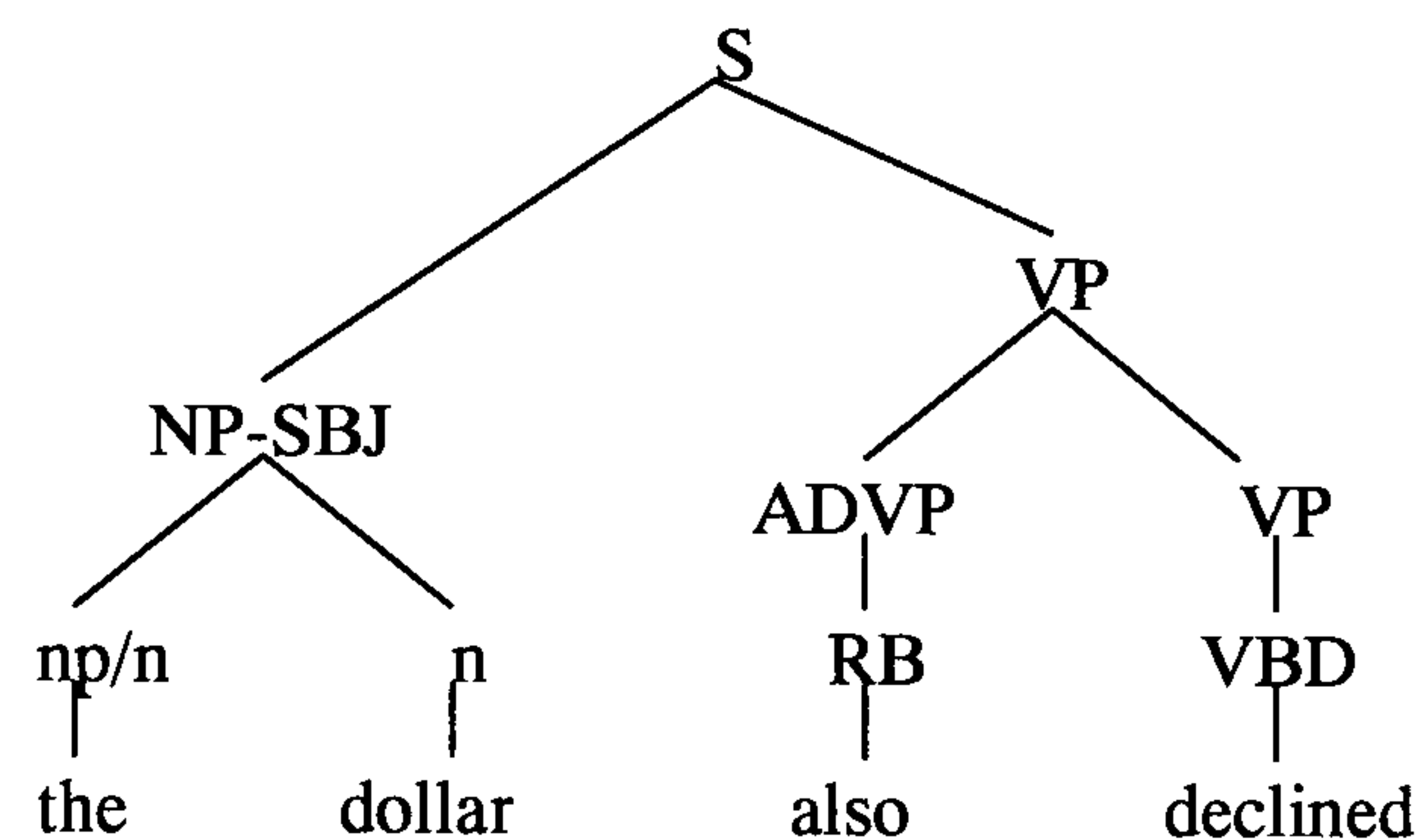


Figure 10.7: Example of the output of Stage 2

something fulfilling the determiner role, then the tag should be translated to the CG category *n*, otherwise it should be translated as an *np*.

The algorithm for applying the set of context-based rules is a simple matching process throughout the treebank. Figure 10.7 shows the output from this stage on an example.

**Stage 3: Structural Heuristic** In this stage, the system uses further knowledge to attempt to inform the translation process. Where words have not been translated, the system annotates the subtree with the head, complements and adjuncts using a modified version of Collins' heuristics [44].

Further categories can now be obtained. For example, if the head of the subtree requires an *np* category to its right as its first complement and there is a word marked as a complement in this position, then it can be translated as an *np*. Alternatively, if the head category is unknown, but it is verbal according to the Penn Treebank label then looking at the categories of the complements can determine the type of verb it is, e.g. no complements following a verb indicates a CG category *s\np*. Figure 10.8 shows the effects of this stage on the example.

**Stage 4: Category Generation** In the final stage each lexical category that has not

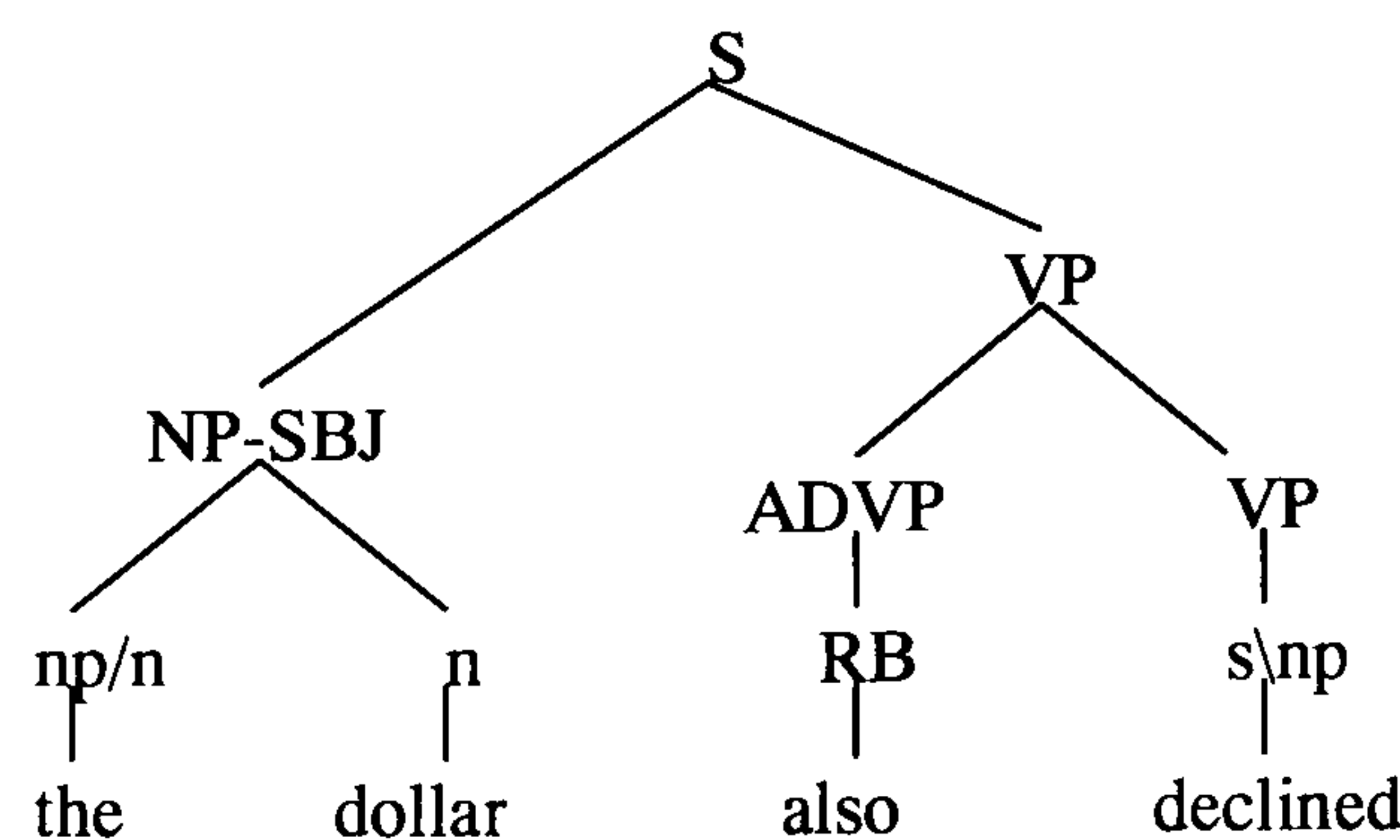


Figure 10.8: Example of the output of Stage 3



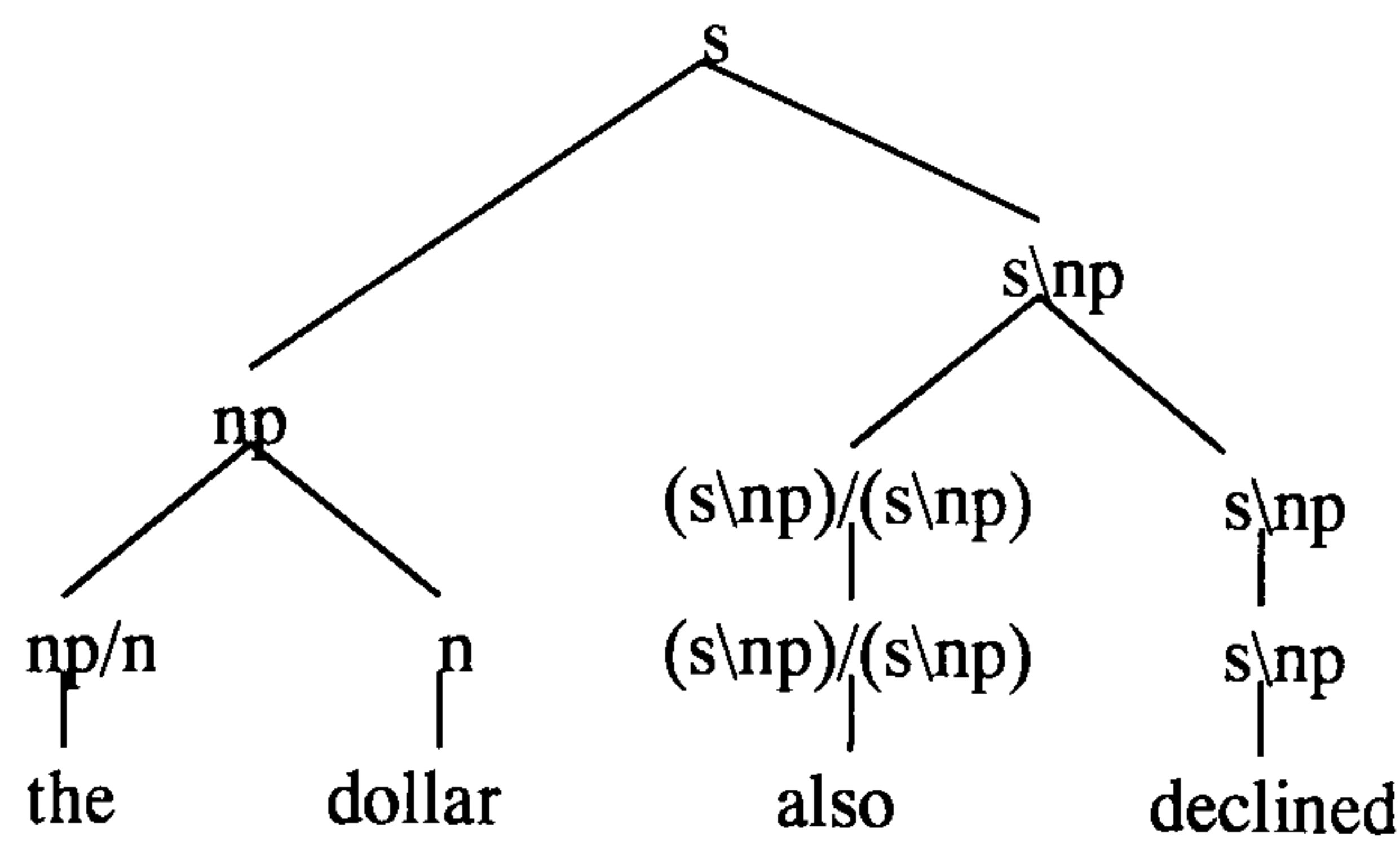


Figure 10.9: Example of the output of Stage 4

been annotated is given a variable for a category. The tree is then traversed bottom-up instantiating these categories by using head, complement and adjunct annotation and the already annotated categories. The building of head and adjunct categories follows the same process described for the top-down algorithm. Complements either gain their categories through this process or have already had them assigned. Figure 10.9 shows the final output.

This approach has two main advantages. Firstly, the user has control over the type of CG to which the treebank is translated, due to the use of predefined categories for predefined contexts. Secondly, the bottom-up approach ensures that translation errors are not propagated seriously through the tree.

A further advantage exists that has not, as yet, been fully investigated. The system, due to its multi-pass nature, has the potential for translations to clash. Experience has shown that this occurs when there is an annotation error, so the system can be used to highlight these and can also provide some level of self-correction. This has not been investigated in detail, but the current approach, which gives satisfactory results, is to assume the head category is correct and adjust complements and adjuncts accordingly. In future, a simple correction scheme could easily be added to produce a self-correcting translator.

The main weakness of the system is the reliance upon the head/complement/adjunct annotating heuristics, which were not designed to be used with a CG.

The system also returns some categories with variables. This is due in part to the heuristics and in part to the small number of rules currently used in the early stages of the translation process. Most of the problem categories could be dealt with by the addition of a few more rules in stages 2 and 3.

#### 10.4.4 Results

Here we provide similar evaluation of the systems as others [67, 145] for easy comparison. Both systems were used to translate C1 and C2. C2 is used for determining the coverage of the grammar used by the two systems. Both systems, at times, failed to translate examples (frequently due to annotation error in the original treebank). The top-down system failed on 60 and 15 examples from C1 and C2 respectively. The bottom-up system failed on 66 and 15 examples from C1 and C2 respectively.

Table 10.1 describes the type of categories used to translate C1 and the size of the lexicons generated. Categories with variables in were ignored, as they could usually be unified with an already existing category. With this in mind, the bottom-up algorithm extracted a more

	Top-down	Bottom-up
No. of cats	167	106
Lexicon size	15887	15136
Ave. cats/word	1.31	1.25
Ave. cat size	8.02	5.12

Table 10.1: Table of category and lexicon information on the translated corpora

Freq. Range	Number of Categories	
	Top-down	Bottom-up
$1 \leq f \leq 1$	42	29
$2 \leq f \leq 10$	61	34
$11 \leq f \leq 20$	14	9
$21 \leq f \leq 100$	24	11
$101 \leq f \leq 1000$	17	13
$1001 \leq f \leq 5000$	7	7
$5001 \leq f \leq 10000$	1	2
$10001 \leq f \leq 12000$	0	1
$12001 \leq f \leq 15000$	1	0

Table 10.2: Table of the category frequencies for both approaches

compact lexicon. The average category sizes (the number of slash operators in categories) are interesting, as they indicate the profligacy of the top-down algorithm in creating unwieldy categories, whereas the bottom-up approach uses smaller and, on inspection, more plausible categories. These results seem, in part, to vindicate the choice of a controlled bottom-up approach.

Tables 10.2 and 10.3 present the results for both systems for the frequency distribution of categories (i.e. the number of categories that appeared with a particular frequency) and the frequency distribution of the number of categories for a word (i.e. the number of words that had a particular number of categories). The trends for both systems are similar. There are a large number of categories that appear very infrequently, these tend to be the larger, generated categories and often fit unusual circumstances, e.g. misannotation of the treebank, or mistakes in the use of the heuristics. The bottom-up approach has many fewer of these categories, indicating the problem of propagating of errors down the tree with the top-down approach. There are also a few exceptionally frequent categories, these are noun phrases, nouns, and some of the common verbs.

The number of categories per word is similar, suggesting the approaches are similar in their ability to produce the variety of categories required for words.

While these figures give some indication of the quality and compactness of the translation, it is useful to determine the coverage of the lexicon extracted from C1 by comparing it with a lexicon extracted from C2 and so determine the quality and generality of the lexicon that has been produced in the translation. Table 10.4 shows the comparison. Here *entry* means the C1 lexicon contains an entry the same as the C2 entry. *kwkc* means that the entry from C2 is not in C1, but both the word and the category are known. *kwuc* means the word is in the C1 lexicon, but the category is not. Finally, *uw* indicates that the word is not in C1.



Freq. Range	Word frequency	
	Top-down	Bottom-up
f=1	10486	10377
f=2	1263	1264
f=3	264	264
f=4	86	86
$5 \leq f \leq 9$	100	100
$10 \leq f \leq 14$	20	20
$15 \leq f \leq 24$	10	10
$25 \leq f \leq 30$	2	2

Table 10.3: Frequencies of words appearing in a frequency range of number of categories

	Top-down	Bottom-up
Categories	98	65
New categories	4	0
entry %	37.29	48.31
kwkc %	10.55	11.09
kwuc %	11.46	0
uw %	40.70	40.60

Table 10.4: Table comparing the coverage of the two approaches

Despite a smaller lexicon and a smaller number of categories, the bottom-up system gives better coverage. Note especially that there are no unknown categories with the bottom-up approach and that the percentage of exact entries is much higher.

#### 10.4.5 Corpus Translation Conclusions

The system presented provides a useful and reasonably accurate method for translating the annotation of the Penn Treebank into a CG annotation. Comparisons with an alternative approach suggest that the increase of control provided by the system leads to a more accurate and compact translation, which is more linguistically plausible. Most importantly, the system is flexible enough to allow the user to annotate corpora with the kind of CG they are interested in, which is vital when it is to be used for evaluation.

It would be useful to expand the systems to work on the full treebank, i.e. including sentences with movement (see Hockenmaier *et al* [67] for discussion of a possible method). The correcting of the annotation of the treebank during translation should also be investigated further.

The system has been applied to the corpora used in the experiments for both training and testing to give a kind of gold standard against which to compare lexical labelling in particular

### 10.5 Evaluation Conclusions

The issue of evaluation for unsupervised learning systems like CLL is a complicated one, as there is little that can be directly compared or measured to indicate how well a system is

performing. For the experiments with CLL, a number of metrics have been proposed that provide strong evidence of different varieties as to how well the lexicon and the parsed corpus are being built. The metrics look at the amount of lexical compression that has occurred; the degree of ambiguity and both labelling and bracketing measures for parsing using the learned lexicons on both training and test data.

A large variety of other metrics could be used to evaluate the results, as there are various in the literature (see Chapter 4). However, this small set appears to provide a useful balance of metrics, as well as allowing some comparison with other systems. The results of applying these metrics for the various experiments described in Chapter 9 are presented in the next chapter.



# Chapter 11

## Results

In this chapter, the results of using the evaluation techniques described in Chapter 10 upon the output of the experiments described in Chapter 9 are presented. Some of these results have already been published [134, 135, 136, 138, 137]. The early results are described first in Section 11.1. These results were used to develop CLL and led to the later work on the corpora PC1–PC4, which were extracted from the Penn Treebank. Each of these corpora has a section to itself below (Sections 11.2–11.5). The sections are laid out in a similar way for ease of comparison between corpora. Some relevant comparisons between results on different corpora are compiled in Section 11.6. This is followed, in Section 11.7, by some comparisons between CLL and some of the alternative systems described in Chapter 4. Finally, some general conclusions about the results are presented in Section 11.8.

### 11.1 Initial Results

The results in this section have been presented in papers on the earlier version of the CLL system [134, 135, 136]. They are essentially the results from experiments with CLL upon the generated corpora GC1, GC2, LLL and LLL(M). Chapter 8 contains a full description of these corpora.

The hand-calculated lexical accuracy and parse accuracy on each corpus is presented in Table 11.1, along with the times that each experiment took. As stated in Chapter 10, the lexical accuracy is the percentage of linguistically plausible lexical entries and the parse accuracy is the percentage of examples which are assigned a plausibly labelled and bracketed parse (quite a strict measure).

The results for the first two corpora are extremely encouraging with 100% accuracy in both measures. While these experiments are on relatively simple corpora, these results strongly suggest the approach can be effective. Note that any experiment on corpus 2 without the closed-class words being set did not terminate in a reasonable time, as the sentences in that corpus are significantly longer and each word may be a large number of categories. It was therefore clear that setting the closed-class words greatly increases speed. This also indicated that it was important to consider methods of relieving the strain on the parser if the approach was to be useful on more complex corpora.

The results with the LLL corpus are also encouraging in part. Lexical accuracies of

Corpus	Closed-Class Words Preset?	Lexical Accuracy (%)	Parse Accuracy (%)	Execution Time (s)
GC1	×	100	100	5297
GC1	✓	100	100	625
GC2	✓	100	100	10524
GC2	×	×	×	×
LLL	×	14.7	0.6	164151
LLL	✓	77.7	58.9	361
LLL(M)	✓	73.2	28.5	182284

Table 11.1: Initial lexical and parse accuracy results

77.7% on the declarative sentences (LLL) and 73.2% on the whole corpus (LLL(M)) are good results, especially considering that the grammar is not designed to cover sentences containing movement and there are many of these in LLL(M). The success is due to the learner parsing the majority of the phrases within examples correctly, with incorrect analysis of the parts that the grammar does not cover, e.g. the question word at the start of a sentence. Clearly, a grammar with greater coverage could cause the results to improve.

The results for parse accuracy, both in the training and test sets, do not suggest that the system is very successful in providing parses. However, the figures do not show the full picture. Practically all the sentences are mostly parsed correctly. Very few of the errors could actually be handled correctly by the grammar with which the system was supplied. In terms of less stringent measures of parse accuracy, e.g. bracket crossing, the performance would be much higher. This is indicated by the much higher parse accuracy measure for the declarative sentences, 58.9%. On this set of sentences, which are much more likely to be covered by the CG categories, the learner performs nearly 30% better. Obviously, in the longer term, a grammar with wider coverage would possibly provide the desired results for this more stringent measure.

It should be noted that the poor results obtained with the LLL corpus without using the closed-class words is due to the severity of the sparse data problem within this corpus. Table 11.2 shows predictably good results for parsing the test sets with the learned lexicons. The timings presented give some idea of the scale of the experiments conducted. As the complexity of the problem increases, it is clear that the time taken by the learner becomes significant. It may be interesting to note that this was the motivation for the efficient Scheme implementation of the probabilistic CKY algorithm discussed in Chapters 5 and 6, which replaced the Prolog chart parser used in this version of CLL. This enabled the experiments on the Penn Treebank [88], a much larger corpus, to run within a reasonable time.

These early results were very encouraging with respect to pursuing the approach that has been outlined in the preceding chapters. The indication is that, upon toy problems, the intuitions behind CLL were accurate and that the system could be used for learning useful syntactic information. However, the question remained as to whether CLL would scale up to more realistic problems.



Corpus	Closed-Class	Parse Accuracy (%)
GC1	×	100
GC1	✓	100
GC2	×	100
GC2	✓	100
LLL(M)	✓	37.2

Table 11.2: Unseen example parsing accuracy

## 11.2 Results for PC1

The next set of experiments was completed on the rather roughly extracted examples from the Penn Treebank contained in PC1. Table 11.3 contains a summary of the results for the set of experiments (see Table 9.1 on Page 158 for a summary of these experiments) carried out using this corpus. In this table and those for the other experiments with the Penn Treebank, all results are presented to two decimal places (where relevant).

Experiment	Lexicon Size	Average Ambiguity	Train		Test	
			Crossing Brackets	Lexical Accuracy	Crossing Brackets	Lexical Accuracy
1	11,405	1.21	5.78	43.26	5.23	44.31
2	12,970	1.24	5.55	49.01	4.93	49.21
3	11,288	1.20	5.66	44.29	5.68	45.09
4	12,851	1.22	5.25	50.34	5.11	50.41
5	11,461	1.22	6.02	42.12	5.88	43.39
6	11,429	1.22	5.41	43.95	5.16	44.72
7	11,399	1.21	5.8	43.07	5.34	44.18
8	11,413	1.22	5.50	44.08	5.20	44.80
9	11,363	1.21	5.67	43.30	5.38	44.40
10	12,683	1.35	6.78	43.91	6.55	44.28

Table 11.3: A summary of the results using the PC1 corpus

A number of figures are needed for a useful comparison. Firstly, with respect to the lexicon size and ambiguity it should be noted that the values for the translated (gold standard) corpus were 14,370 and 1.24 respectively. With respect to the crossing-brackets rate, Table 11.4 contains the baselines against which the learned values can be compared. Each of the baseline heuristics is applied to both the training and the test data. Finally, the baselines for lexical accuracy are given in Table 11.5. Firstly, there are the two random baselines, which are based on the number of categories and thus whether CATDB2 or CATDB3 is used in the experiment (only experiment 5 used CATDB3). The random baselines, as they are in fact expected values, are the same for both the training and the test data. Secondly, the baseline of assigning all words the np category is presented. As this baseline requires calculation of the actual results, the table shows that the values are slightly different for training and test data.

With respect to the lexicon size, the results range from 11,288 to 12,851. These sizes are about 2000 entries smaller than the gold standard lexicon. The main reason for this is that

Baseline	Train	Test
Right Branching	4.49	4.29
Left Branching	14.86	14.45
Random	9.38	9.22

Table 11.4: The baseline crossing-brackets rate results for PC1

Baseline	Train	Test
Random CATDB2	3.33%	3.33%
Random CATDB3	2.22%	2.22%
All np	25.51%	25.63%

Table 11.5: The baseline lexical-accuracy values for PC1

the learner fails to parse a number of the examples in each experiment and so lexical entries are not available from these examples. Another reason is that the translated lexicon has more categories available to it. However, comparison between experiments seems to suggest that larger lexicons are providing better results with the other metrics (see experiments 2 and 4). This would suggest that the lexicon is currently being compressed too much.

The ambiguity measure is more useful. In general, while the ambiguities are slightly lower than that of the gold standard corpus, they are fairly similar, with experiment 2 returning the same average ambiguity. The result that stands out with respect to the ambiguity is experiment 10, where the ambiguity is much higher. This is not surprising, as this experiment does not allow noun-phrase joining and so for compound noun-phrases the parser will be forced to build more complex structure, thus giving many nominals a larger number of categories.

The average crossing-bracket rates indicate some success. They are all significantly better than the left-branching and the random baselines, which is encouraging. They are however, in general, slightly lower than the right-branching baseline. This is not surprising given that the right-branching heuristic is such a strong indicator for English, especially given the rather flat annotation of the Penn Treebank which removes a lot of the structure that would be contrary to a purely right-branching heuristic. Hence, it remains positive that a system which had the potential to be both left-branching or right-branching has produced trees that give a crossing-bracket rate much closer to that of a right-branching language. However, it is disappointing that it has not been possible to improve upon the right-branching heuristic.

The baselines for lexical accuracy show that the system is far more effective than simply assigning categories randomly. In general the system is performing about 40% better than this baseline, which is a very large improvement. Experiment 5, which uses CATDB3 and therefore has a random base line of 2.22% shows a similar drop to the baseline because of this larger category set. The results are less spectacular when compared with the np baseline. In this case, performance is around 20% better with CLL, which still shows a large improvement. Therefore, while the labelling results do not necessarily appear to be very high, they are in fact quite a large first step, given the baseline performance.

It should be noted with respect to the training and test sets that the results are slightly



higher for both lexical-accuracy and crossing-brackets metrics for the test set. This is almost certainly due to the slight reduction in the complexity of the test corpus when compared to the training corpus (see Chapter 8).

With respect to the different parameter settings, some conclusions can be drawn. It seems clear from experiments 2 and 4 that the larger initial lexicon produces better results for both the crossing-brackets and the lexical-accuracy metrics. This is because the lexicon provides more information to CLL with respect to the parsing phase in particular. Similarly, it would appear that fixing the closed-class words also improves results, as shown in the results for experiments 3 and 4.

Extending the category database appears to be a mistake, as it results in a fairly large decrease in lexical accuracy and a fairly large increase in the crossing-brackets rate in experiment 5. This is perhaps not surprising, as increasing the size of the category database without increasing the constraints on how it should be used simply increases the potential ambiguity and thus the potential number of parses.

The experiments with different beam values appear to indicate that the use of smaller beams gives better results. Somewhat surprisingly, the results would suggest that the use of compression as well as the probabilistic methods is not useful and that the system should rely solely on the probabilistic methods. An alternative interpretation would be that much larger beams would have to be allowed for the compression metric to have a significant impact.

The two compression metrics interestingly appear to show that compressing the corpus rather than the lexicon provides better results. Experiments 8 and 9 in general give somewhat better results than the equivalent experiments using lexicon compression (1 and 7) for both the crossing-brackets and lexical-accuracy measures. This may seem somewhat unintuitive, as one might expect that the compression of the artefact that is being learned would be better. However, it is perhaps not surprising that an approach which compresses both the artefact and the data (i.e. an MDL-like approach), performs somewhat better.

It is also fairly clear from the results for experiment 10 that the inclusion of the extra rule in the categorial formalism for combining noun phrases is very useful, as the removal of the rule increases the crossing-bracket rate, although it does not change the lexical accuracy very much.

The results of these experiments on PC1 are promising if not spectacular. They suggest that CLL is achieving reasonable results on a difficult problem. The next step is to evaluate the results on a cleaner corpus to determine the extent of the effect of noise on CLL. Hence, the next set of experiments is performed on PC2.

### 11.3 Results for PC2

The results for the experiments (see Table 9.1 on Page 158 for a summary of these experiments) performed with PC2 are presented in Table 11.6. Some of these results were presented by Watkinson and Manandhar [138]. Given that the corpus is somewhat different to PC1, it is necessary to present some of the baseline information with respect to PC2. Hence, the lexicon size and ambiguity of the translated corpus, against which the learned values can be compared, are 15,136 and 1.25 respectively, i.e. PC2 is a slightly more complex corpus.

Experiment	Lexicon Size	Average Ambiguity	Train		Test	
			Crossing Brackets	Lexical Accuracy	Crossing Brackets	Lexical Accuracy
1	12,076	1.21	5.43	44.76	4.70	47.53
2	13,851	1.24	5.61	49.54	4.86	51.89
3	11,905	1.20	5.63	45.16	5.16	47.41
4	13,835	1.24	5.63	49.83	5.01	51.94
5	12,176	1.22	6.19	45.62	5.49	48.16
6	12,070	1.21	5.44	45.36	4.48	47.68
7	12,044	1.21	5.96	44.56	5.32	47.26
8	12,074	1.21	5.48	44.79	4.69	47.48
9	12,072	1.21	5.89	44.47	5.22	47.11
10	13,508	1.36	6.48	47.87	6.08	48.98

Table 11.6: A summary of the results using the PC2 corpus

Table 11.7 contains the crossing-bracket-rate baselines for PC2. Again, these are as expected with the right-branching baseline being very reasonable and the random and left-branching figures being somewhat lower. It is, however, noticeable that these figures are significantly higher than those for PC1, which is another indication that the corpus has increased in complexity even though it is cleaner. As before, the values for the test corpus are somewhat lower. This is again probably due to the slightly lower complexity of the test section of this corpus (see Chapter 8).

Baseline	Train	Test
Right Branching	4.97	4.60
Left Branching	16.74	16.56
Random	10.70	10.60

Table 11.7: The baseline crossing-brackets rate results for PC2

Table 11.8 contains the baselines for lexical accuracy, which for the random approaches are the same, as they are expected values. However, it will be noted that for the baseline where all categories are set to np, it will be noted that results are slightly worse than those for PC1, again indicating that PC2 is somewhat more complex.

Baseline	Train	Test
Random CATDB2	3.33%	3.33%
Random CATDB3	2.22%	2.22%
All np	23.37%	23.89%

Table 11.8: The baseline lexical-accuracy values for PC2

With respect to the lexicon size metric, the experiments give results ranging from 11,905 – 13,851, which, again, is rather smaller than the translated corpus lexicon. This can again be explained mostly by the set of examples that the learner cannot parse. It is interesting to note that the lexicons are bigger than for PC1 and so CLL is at least allowing the learned



lexicons to grow where necessary.

Ambiguity figures, although again lower than the translated corpus, are reasonably close, especially experiments 2 and 4. The differences are almost certainly due to the smaller number of categories available to the learner when compared to those available in the translated system.

The crossing-bracket rates are respectable for the training data. They are generally somewhat lower than the results for the same experiments with PC1 and given the increased values of the baselines, that would suggest an overall improvement of performance on the cleaner data. Again they are significantly better than the left-branching baseline and the random baseline and they are closer to the right-branching baseline (as this is higher for than for right-branching baseline of PC1). The results for the crossing brackets measure are more encouraging still for the test data. The values are in general rather lower than for the same experiments with PC1 and they are much closer to the right-branching baseline, including the value for experiment 6, which is 0.12 below the right-branching baseline. Hence, it has been shown to be possible to improve upon this baseline value with CLL, even though the right-branching information is not included within the learner (right-branching, left-branching or an inconsistent branching would be possible from the set of categories in the category database, although there may be some bias towards a right-branching grammar).

The lexical-accuracy rates are also somewhat improved, especially in the context where the baselines have got somewhat lower. The improvement over the random baselines is now around 45%. Similarly the improvement over the np baseline is around 27%. Results are generally better for the training data than the test data. Almost all the values are improved when compared with the same experiments with PC1 (experiment 4 with the training data being the one exception), with the best values being for experiments 2 and 4 again. These improved results suggest that the use of a cleaner corpus does improve lexical accuracy.

There is an interesting dichotomy here that when the probabilistic constraints are stronger (experiment 6), then the crossing-bracket rate is at its best. However, when the initial information is better (experiments 2 and 4), then the lexical accuracy figures are at their best. It may be interesting in future to determine whether a combination produces the best result.

The pattern with respect to the parameters is fairly similar to the experiments with PC1. However, with respect to the large initial lexicon, while it improves lexical accuracy, the crossing-bracket rate rises (compare experiment 1 to experiments 2 and 4). The results are the same for the fixed-closed-class-word parameter (compare experiments 1 and 3). This would seem to indicate that these constraints can be applied too severely with respect to the bracketing of the sentences, but such information continues to constrain the labelling in a way that adds more information than it detracts.

Again the use of a larger category database reduces the quality of the performance for both the crossing-brackets measure and the lexical-accuracy measure (see experiment 5).

The results with the change in the value of the beam show essentially the same pattern as with PC1. The smaller the value of the beam the better the results (compare experiments 1, 6 and 7).

In these experiments, the different compression metrics appear to have a fairly small effect. Comparing experiments 1 and 8 and 7 and 9 shows that the results are very close

and there is no consistent pattern of one metric being better than the other.

Again the removal of the noun-phrase-joining rule appears to reduce the performance causing a significant increase in lexicon size and ambiguity and poorer result for the crossing-brackets metric. However, the flexibility that removing this rule affords in assigning categories increases the labelling accuracy metric. This is presumably because the structured noun phrases built without the joining rule more closely mirror the labelling in the gold-standard corpus.

## 11.4 Results for PC3

In this section the results of the experiments with the corpus PC3 are presented. PC3 is exactly the same set of examples as PC2, hence the gold standard corpus is the same and the baseline values for the bracketing and the lexical accuracy are also the same. However, PC3 has the noun and verb (NV) annotation, thus allowing a different setting of CLL. The experiments have been performed in two sets. Firstly there are those experiments without any initial lexicon (see Table 9.2 on Page 158 for a summary of these experiments), the results of which are presented in Table 11.9. This does not contain results for experiments 2, 3 and 4, which use alternative initial lexicon settings, as these experiments are not relevant when no initial lexicon is being used. Secondly, the set of results for experiments carried out with initial lexicons (see Table 9.1 on Page 158 for a summary of these experiments) is presented in Table 11.10. These results will be discussed in turn below. Some of the results carried out with this setting have been published by Watkinson and Manandhar [137].

Experiment	Lexicon Size	Average Ambiguity	Train		Test	
			Crossing Brackets	Lexical Accuracy	Crossing Brackets	Lexical Accuracy
1	13,010	1.12	5.11	35.59	4.40	35.48
5	12,590	1.11	5.87	30.58	4.98	31.71
6	12,903	1.11	5.19	37.11	4.38	37.22
7	12,852	1.11	5.99	31.14	5.24	32.05
8	13,009	1.12	5.11	35.59	4.41	35.46
9	12,853	1.11	5.99	31.14	5.27	32.05
10	13,683	1.22	5.11	42.45	4.66	42.10

Table 11.9: A summary of the results using the PC3 corpus without an initial lexicon

Firstly we will consider the experiments that did not use an initial lexicon. The lexicon size results are similar to those of PC2, again these are possibly a little small given the size of the gold standard lexicon, although it is hard to determine if this is due to too much compression, or the lack of parsing examples. It is interesting to note that, in general, more examples were parsed with this corpus than with the unannotated setting, suggesting that this setting is slightly less constrained.

The ambiguity values are rather lower than previous results and quite a lot lower than the gold standard. This does suggest too much compression. It would seem that CLL is too biased towards single lexical entry words with this setting.



Despite this, the crossing brackets figures are much better than those achieved with experiments on either PC1 or PC2. A best of 4.38 for the test set is reasonably well below the right-branching baseline, although none of the crossing-bracket rates are below the right-branching baseline for the training set. However, these are significantly improved results. The values are again much better than those for the random or left-branching baselines and remembering that CLL does not have the right-branching heuristic supplied, the results remain encouraging.

The lexical accuracy results are, however, quite disappointing. Improvement over the random baseline is now down to a little over 30% and similarly, improvement over the np baseline is down to a values of around 8%. While it would appear that the NV annotation allows better structuring of the examples (perhaps not surprisingly, as nouns and verbs are central to sentence structure) it would appear that it allows too much freedom in the labelling for CLL to perform well. The reduction in accuracy is about 10% when compared to the experiments on PC2, which is fairly severe. However, it should perhaps be noted that the removal of the initial lexicon does increase the ambiguity in the problem, as no words have fixed categories.

With respect to the different parameters, the pattern remains similar. The large category set caused a poor performance as before. With respect to the beam, again smaller would appear to be better, with a beam of 1 generally giving the best results.

In these experiments the different types of compression had practically no impact on the result. Comparing experiments 1 and 8 and 7 and 9 show that the results are almost identical. Hence, in this context it would appear that the compression metric has no impact.

Experiment	Lexicon Size	Average Ambiguity	Train		Test	
			Crossing Brackets	Lexical Accuracy	Crossing Brackets	Lexical Accuracy
1	12,069	1.13	5.23	36.36	4.38	36.12
2	13,092	1.13	5.18	36.29	4.55	36.16
3	12,772	1.10	5.39	33.71	4.62	33.90
4	12,794	1.10	5.32	33.72	4.64	33.99
5	12,594	1.11	5.78	30.83	4.95	32.00
6	12,959	1.12	5.17	37.25	4.54	37.17
7	12,884	1.11	5.98	31.20	5.20	31.95
8	13,660	1.20	4.45	47.26	3.78	48.38
9	12,882	1.11	5.95	31.23	5.10	31.99
10	13,722	1.23	5.12	41.78	4.69	41.77

Table 11.10: A summary of the results using the PC3 corpus with an initial lexicon

Secondly, we will consider the experiments performed with an initial lexicon. Again the lexicon sizes appear to be somewhat small, although the high of 13,660 is approaching the right size. The range within the experiments is fairly large with sizes from 12,069 to 13,660. In general a larger lexicon is good, as the best performing settings (experiments 2, 6 and 8) also have the three largest lexicons.

Again ambiguity appears to be rather too low, although it is in general higher than the results from experiments on PC3 without the initial lexicon. Only experiment 8 appears to



have achieved an ambiguity of the right kind of order, although it remains a little low. It is interesting that experiment 8 also gives the best labelling and crossing-bracket values as well.

In general, the results for crossing brackets are slightly better than without the initial lexicon. The labelling frequencies appear to be fairly similar and are still low. There is, of course, one set of results that stands out, which is experiment 8. This returns the best lexical-labelling figures with this corpus and also produces crossing-bracket rates well below even the best baseline (the right-branching baseline) for both training and test data. A crossing bracket rate of 3.78 for the test data is approaching the results of more supervised systems and even some parsers. This is a very encouraging result, although it is in the context of the most supervised setting that has currently been investigated for CLL.

The lexical accuracy remains lower than experiments performed with CLL on PC2, giving only slightly higher values (except for experiment 8) than the experiment on PC3 *without* an initial lexicon, which suggests that the NV annotation actually has a negative effect on the lexical accuracy. This would indicate that further work needs to be pursued to determine what combination of knowledge is required to improve the labelling. It is possible that some conservative extension of the set of lexical categories might be valuable.

Interestingly, the pattern of results with respect to the different parameters seems to have changed with these experiments. Using a bigger initial lexicon reduces performance, as does fixing the closed-class lexicon. This results in experiment 4 being somewhat worse for both bracketing and labelling when compared with experiment 1. This would suggest that the two types of knowledge (the NV annotation and the initial lexicon) do not interact well.

However, the larger category database still causes a drop in performance and the smaller the beam the better the performance in general, thus conforming to the previous patterns with respect to these parameters.

In these experiments, compression of the corpus provides an improvement in performance over lexicon compression, especially in experiment 8. It would appear that, in experiment 9, the extra freedom afforded by a beam of 4 negates the effects of the constraints imposed.

The removal of the noun-phrase-joining rule has again produced mixed results. The lexicon size and the ambiguity are much higher. The ambiguity in particular is implausibly high. Again the crossing-bracket rates are higher than when the rule is included. However, the lexical accuracy has again increased. Again it would appear that the removal of the rule allows categories to be assigned to words in compound noun phrases that are more akin to the categories assigned to those words in the gold-standard corpus. However, it would also appear that these more complex structures are more likely to allow incorrect structure to be proposed.

In general then, the NV setting has provided the best results with respect to the crossing-bracket measure. This is perhaps not surprising as the annotation does provide information about the overall sentence structure. This approach has provided some results that have significantly improved upon all the bracketing baselines. However, it needs to be remembered that this has occurred for the most supervised setting of CLL.

With respect to labelling, while there were respectable results when the initial lexicon was included, the results are in general much lower than those for experiments with PC2.



This is again not surprising, as the ambiguity with respect to labelling is intuitively lower when the initial lexicon is used. However, the introduction of an initial lexicon did not improve the results very much.

The results are encouraging on these corpora of smaller examples and so experiments were performed on a larger corpus, PC4.

11.5 Results for PC4

In this section, the results from the two experiments performed on the larger examples corpus (i.e.  $\leq 25$  word examples), PC4, are presented (see Table 9.1 on Page 158 for a summary of these experiments). Only two experiments have been completed with this corpus, as the experiments are some of the most recent and they also take significantly longer to perform due to the increased average example length. The appropriate initial lexicons and the NV annotation were supplied to CLL in these experiments.

Experiment	Lexicon Size	Average Ambiguity	Train		Test	
			Crossing Brackets	Lexical Accuracy	Crossing Brackets	Lexical Accuracy
1	15,401	1.13	11.06	36.19	9.62	36.32
4	15,962	1.17	10.95	39.61	9.57	39.93

Table 11.11: A summary of the results using the PC4 corpus

Again a number of figures are needed for a useful comparison. Firstly, the lexicon size and ambiguity for the translated corpus were 17,876 and 1.27 respectively. It would appear from these experiments that the lexicon was compressed too much. Table 11.12 contains the crossing-bracket-rate baselines against which the learned values can be compared. Each of the baseline heuristics is applied to both the training and the test data. As before, the baselines for lexical accuracy are also supplied (in Table 11.13). As no experiment has been performed with CATDB3, this baseline is not included.

Baseline	Train	Test
Right Branching	9.95	8.79
Left Branching	29.33	26.43
Random	18.10	16.25

Table 11.12: The baseline crossing-brackets rate results for PC4 without initial lexicon

Baseline	Train	Test
Random CATDB2	3.33%	3.33%
All np	22.37%	23.06%

Table 11.13: The baseline lexical-accuracy values for PC4

The results for lexicon size and ambiguity follow a similar pattern to those above. They are both a little low, perhaps especially the ambiguity. The crossing bracket values are

disappointing, as they are really quite high. This raises some concerns about the viability of extending CLL to work on large scale corpora. It will be necessary to improve the grammatical coverage and accuracy to achieve better results. Having said this, the results are still significantly better than the left-branching and the random baselines, suggesting that significant learning has occurred. The results can in this sense be seen as an extension of the results for PC3. It would be interesting to do some further experiments using the corpus-compression technique and see if equivalent improvements could be achieved.

The lexical accuracy has not decreased significantly from that of the same experiments for PC3, which is interesting, especially as this is a feature mirrored by the baseline. It would appear that lexical accuracy, unlike crossing-bracket rate, is not connected to sentence length. The lack of connection between the crossing-bracket rate and the lexical accuracy needs to be investigated further. It does indicate that parts of the trees are correctly formed, but that small structural mistakes can have a big impact on the crossing bracket rate.

The results show that experiment 4, with a larger lexicon and the fixed closed-class words provided better results for both the crossing-bracket and the lexical-accuracy metrics. In future, further parameters need to be investigated.

## 11.6 Cross-Corpora Comparisons

In this section, I will aim to unify some of the conclusions about the parameters that can be used with CLL. Each of the parameters will be discussed in turn below. Following that, some examples of how the lexicon grows and how the crossing-brackets and lexical-accuracy measures change as more examples of the corpus are dealt with are shown.

### 11.6.1 The Corpus

It seems clear that the corpus has some impact on the effectiveness of the learner. PC1, although seemingly being a simpler corpus, produces worse results because it contains noise. The inclusion of utterances that are not sentences and some stray punctuation elements do appear to cause some problems.

The results with PC2 are much better, even though it is a more complex corpus. The experiments with PC2 provide the best results for lexical accuracy of all the experiments. This suggests that no annotation, but the use of initial lexicons is a better way of building a lexicon which provides the right annotation. The reason for this is probably that the bias provided by the initial lexicon is towards the accurate labelling of words.

However, PC3 provides probably the best overall results. It is only PC3 that produced crossing-bracket rates that are below the right-branching baseline for both the training and testing corpora. This would be more encouraging if the lexical-accuracy scores were better. Unfortunately these values appear to be 10% down on the results achieved with PC2. However, there were some results that returned lexical labellings of just below 50%.

These results are perhaps not surprising, as the NV annotation of PC3 provides a stronger structural indication than the initial lexicons used in the experiments on PC1 and PC2. However, it does intuitively provide less specific labelling information. Perhaps the more surprising result is that, while using both the NV annotation and an initial lexicon does retain



the good crossing-brackets rate achieved without an initial lexicon, it does not improve the lexical accuracy much. This perhaps suggests something of a clash between the knowledge sources, i.e. the initial lexicon and the NV annotation.

Overall, the results from these experiments are positive. The evaluation appears to suggest that a significant amount of lexical and structural information has been captured in the lexicon. However, there appears to be work to be done in improving the interaction of knowledge sources to improve the results. It did, however, seem worth extending the experiments to a corpus with larger examples.

As yet, the experiments on PC4 are not conclusive, but they do appear to be a little disappointing with respect to the crossing-bracket rate. It would appear that the increase in crossing-bracket rate is going to be related to example length. The encouraging factor here is that the relation between the crossing-bracket rate of CLL and the baseline rates appears to remain about the same. However, with respect to the lexical accuracy, the results indicate a reasonably similar performance here when compared with the experiments on corpora of shorter examples, but these are still somewhat low.

### 11.6.2 The Initial Lexicon

The use of different initial lexicons produced conflicting results. With PC1 both the crossing-brackets measure and the lexical-accuracy measure improved with a larger initial lexicon. However, with PC2, while the larger lexicon improved lexical accuracy to its highest level, it also caused an increase in the crossing-bracket rate. With PC3, the impact of the larger lexicon was the most confusing. The crossing-bracket rate was lower for the training and higher for the test data. The lexical accuracy was also lower for the training and higher for the test data.

The implication would appear to be that the size of the initial lexicon does have an impact, particularly where there are less other constraints. This leads to the improvement with PC1 and PC2, where there is no other bias from annotation. It is not surprising that the improvement is most marked in the lexical accuracy, as this is what the initial lexicon most directly affects.

However, where other bias is supplied, like the NV annotation, the initial lexicon may not be very useful, as it does not cause very much improvement and can even degrade performance. This would suggest that some other knowledge is needed to bias CLL to improve performance.

### 11.6.3 The Fixed Closed-Class Lexicon

Fixing the closed-class lexicon, while occasionally improving the performance of the system (e.g. with PC1), seems in general to have a negative effect. It would appear that it is too restrictive to assume that no other words in the corpus require a closed-class category.

This can be explained in two ways, however. The first would be that there are other words in the corpus that really do use these categories (which would in part explain the increase in lexical accuracy when they are allowed to take these categories). In the case of the smaller closed-class lexicon (CCW1), this will be the case, as many of the closed-class

words are not included. However, the larger closed-class lexicon (CCW2), was actually built using the Penn Treebank and so should contain all of the words that should take these categories (assuming they are annotated correctly in the Penn Treebank and that it is only the parts-of-speech considered that should be assigned these categories). Hence, in the cases where CCW2 is used, this explanation is less plausible. The second explanation is that the use of more categories for unknown words increases the flexibility of the parsing process in CLL, which allows better approximations to the correct parses. Hence, this result may be an indication of the need for more categories to be allowed, i.e. broader coverage in the category database. Further work needs to be carried out to test this.

#### 11.6.4 The Category Database

Part of testing whether a broader-coverage category database would be useful is provided by using the large category database extracted from the translated corpus. This has led to uniformly bad results for both the lexical accuracy and the crossing-bracket metrics.

While this is an indication that larger category databases are not the answer, the results perhaps need to be considered a little more carefully with respect to how the large category database is built. It may be the case that simply extracting the most common categories from an (less than perfect) annotation of the Penn Treebank is not the best way to extend the category database. Instead, using a more linguistically motivated approach to adding a few categories may improve things significantly. Alternatively, it may be necessary to provide some prior weighting for categories, as it may be that the addition of extra categories, while improving some analyses where they are needed, also allows a large variety of other analyses for examples for which they are not needed.

The category database and its use will require some further analysis in the future if the system is to perform better and especially if it is to perform well on broader corpora including movement.

#### 11.6.5 The Beam

In general, as has been noted, the smaller the beam, the better the performance. The beam of four produced consistently worse results than the beam of two. The beam of one, an unusual setting because it does not allow the compression metric to have any effect, commonly (although not quite consistently) produced better results than the beam of two. However, this is only with the lexicon compression metric. The later experiments with corpus compression were rather different, producing better results than the beam of one with lexicon compression in some cases.

There appear to be two main conclusions to draw with respect to the value of the beam. Firstly, the probabilistic constraints of the best parses are important. The more weight given to them, the better the results. Secondly, however, this needs to be tempered by their interaction with other constraints, in particular the type of compression.



### 11.6.6 The Compression Metric

The compression metric was surprisingly important. Initially it was assumed that it would be correct to compress the lexicon, as this was what was to be learned and what would be stored in the human brain. The results however suggest that the frequency information in the lexicon needs to be taken into account. Using corpus compression, which allows this, generally produced similar or better results. Such results would appear to mean that focusing on the most frequent information is a better approach.

In future, it would appear sensible to try alternative methods of compression, especially given the impact of small changes in the experiments here. In particular, it would be interesting to build a rigorous MDL approach, which would combine both the current compression metrics.

### 11.6.7 The NP Rule

The removal of the noun-phrase-joining rule had mixed results. The ambiguity was commonly raised to an implausible value, presumably due to the increase in entries for words involved in compound noun phrases. The increase in categories allowed an increase in possible structure, which appears to have caused a decrease in the quality of the learned lexicons with respect to their ability to describe the appropriate structures for sentences. The crossing-bracket rate was universally worse without the rule.

However, the lexical accuracy was often improved with the rule. This is due to the fact that the gold standard corpus includes the structure for the compound noun-phrases. The improvement in the lexical accuracy suggests that CLL proposed the same internal structures for these phrases as the gold-standard corpus.

### 11.6.8 Experiment Length

In general, timing results are often not especially useful, as they are dependent on too many factors that are not related to the actual algorithms, or even to the implementation of those algorithms. In the case of CLL, the time taken to complete experiments has been difficult to calculate. The experiments on the short-example corpora take approximately a week to perform on a 750MHz PC running Linux and the long-example corpora take approximately two weeks, although obviously time varies depending on the setting of CLL amongst other things. Unfortunately more precise times are hard to determine, as the network that the experiments were being run on caused there to be a need to restart experiments part way through and thus timings were not retained.

While we are left with some intuitions taken from experience doing the experiments, for example, it is clear that the experiments that used longer examples took longer and the experiments took longer the larger the beam of the parser. Both of these intuitions are, in fact, fairly obvious results of the algorithm. Unfortunately, little more can currently be said on this issue. However, the overriding result is that the experiments can be completed in an acceptable time. In future, it may be useful to be able to investigate the impact of the parameters more empirically, as this may help in determining areas where efficiency could be improved.

### 11.6.9 Experiment Progression

In this section, I will present some comparisons of the way the lexicons grew, the way the crossing-bracket rates changed and how the lexical-accuracy values changed as the experiments progressed. Two issues need to be remembered when considering this data. Firstly, as the experiments progress CLL is hopefully building a better and better lexicon. Secondly, because the corpus is ordered on example length, the most difficult sentences occur later in the corpus. Figure 11.1 shows how the example length changes for PC1, PC2 (and PC3 as it contains the same examples as PC2) and PC4. Better lexicons will cause performance to improve. Longer examples will cause performance to drop. Hence, there will be a trade-off between the two that will lead to the actual results.

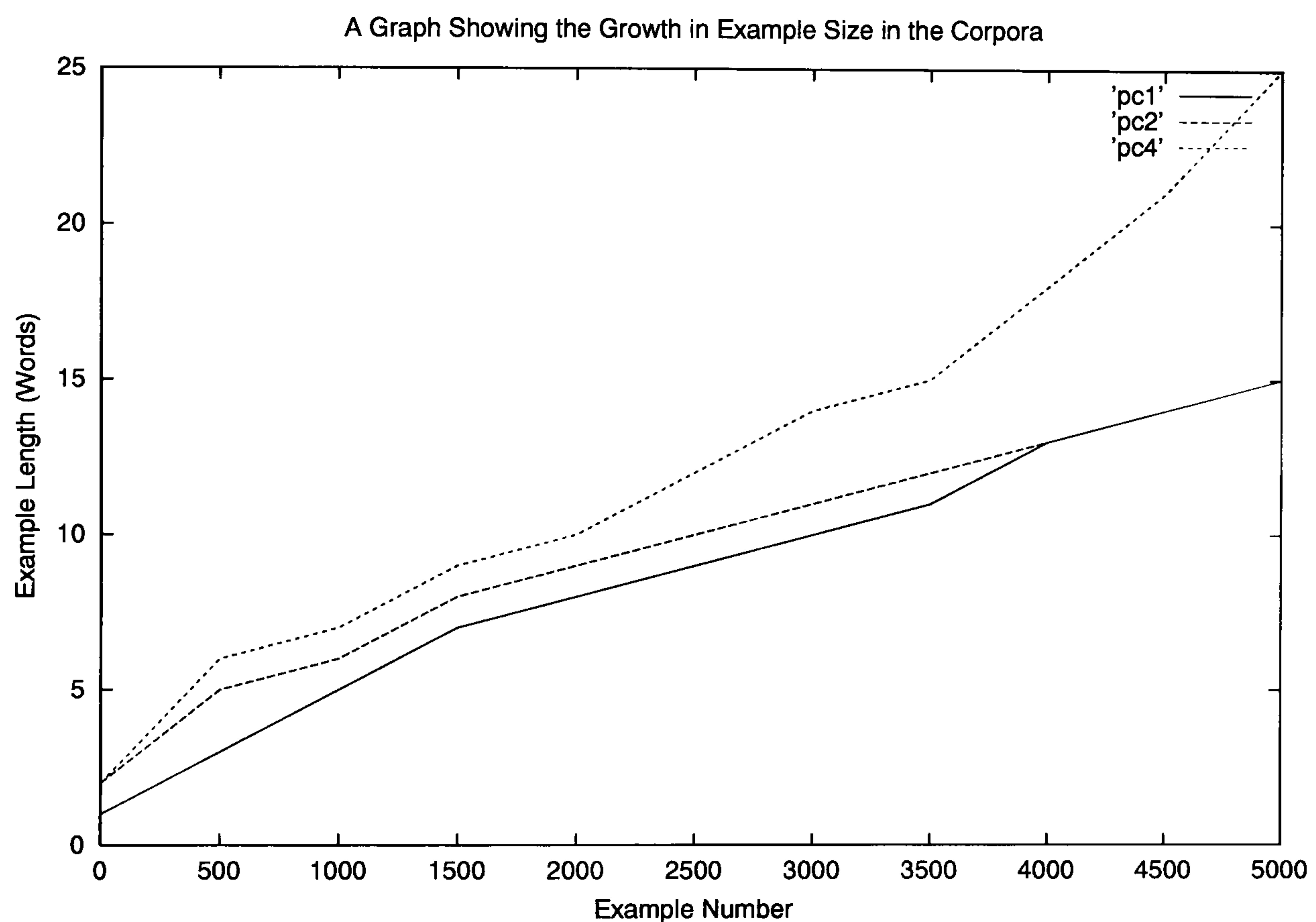


Figure 11.1: A graph showing the increase in example size through the corpora

The first progression graph is shown in Figure 11.2. It shows how the lexicons grow for experiment 1 using the different corpora. The experiment for PC3 is the one without any initial lexicon. It seems from this graph that in all cases the lexicon grows in a somewhat linear fashion. One thing that is very clear is that no grammar (i.e. lexicon in this case) has been converged upon. This is in part due to the large number of new words that occur as CLL progresses through the corpus. However, it also indicates that new categories are being learned for words that have already been seen. This graph would suggest that CLL would perform much better if it could be applied to significantly larger corpora. However, currently that would probably be too expensive computationally, in particular with respect to time.

All the corpora follow the same pattern, although PC1 produces a consistently smaller lexicon than PC2 or PC3. This is due to the set of one-word examples at the start of the corpus, which are not allowed in PC2 and PC3, because they are not sentences. PC2 and PC3 diverge slightly, suggesting that PC2 is perhaps converging slightly faster.

The graph of how crossing-bracket rate varies over the number of examples processed is



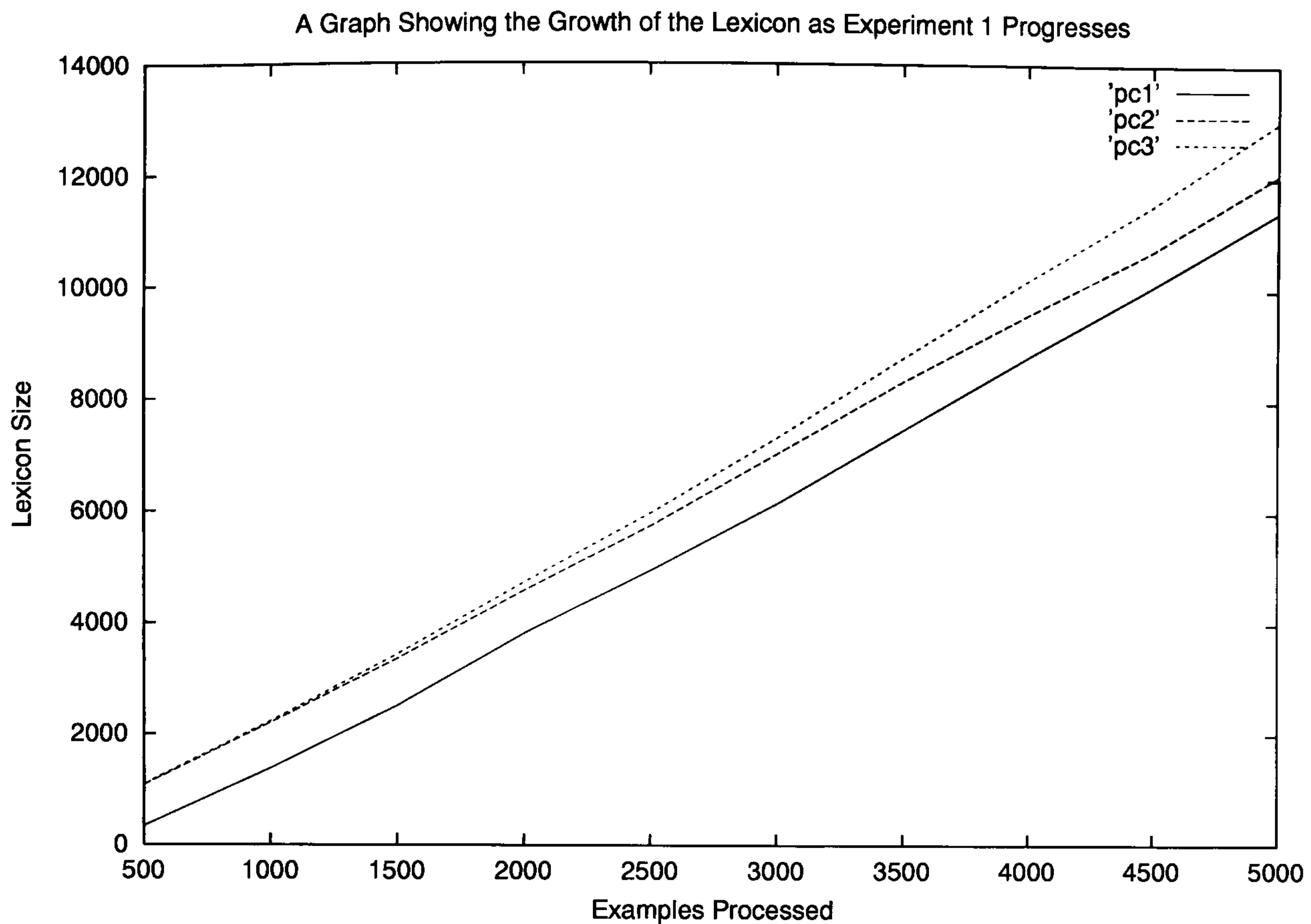


Figure 11.2: A graph showing the lexicon growth for experiment 1

shown in Figure 11.3. This indicates an unpleasant growth of average crossing-bracket rate as the example length increases (note this is also shown by the experiments using PC4). In one sense, the increase is not surprising, as the examples in the corpus are ordered by increasing number of words. As the number of possible brackets is exponential in the number of words, then if CLL was not learning anything the graph should show a severe exponential curve. Hence, the fact that the graph has a fairly shallow curve indicates that CLL is successful to some extent. The rate of lexicon growth (shown in Figure 11.2) also suggests something of an issue with respect to this curve. Ideally, although the lexicon will continue to grow with new word (especially new names), if the lexicon (i.e. the grammar) is converging then the lexicon growth curve should flatten. This is not the case and so if new words are continually appearing and the lexicon is not converging yet then the crossing bracket rate is unlikely to drop. This graph, therefore, also motivates the need to do longer experiments over much more data to determine if extra data will allow the lexicon to converge. Unfortunately, with the current implementation of CLL, this is impractical due to the time required for the experiments. Once again, there is motivation for improving the efficiency of CLL by either providing an alternative to reparsing, or provide a faster way of parsing.

Again, all the experiments with all the corpora follow the same pattern. PC1 shows a sharper degradation in performance than the other two corpora, which again indicates that CLL is somewhat sensitive to noise. The experiment with PC2 remains consistently worse than PC3, which illustrates the general point that the experiments with PC3 perform better in terms of structural learning.

Finally, Figure 11.4 shows the change in lexical accuracy as examples are processed by CLL. It is clear that the performance is degrading throughout. The unusual spike in the PC1 corpus experiment is again due to the set of single word examples at the start of the

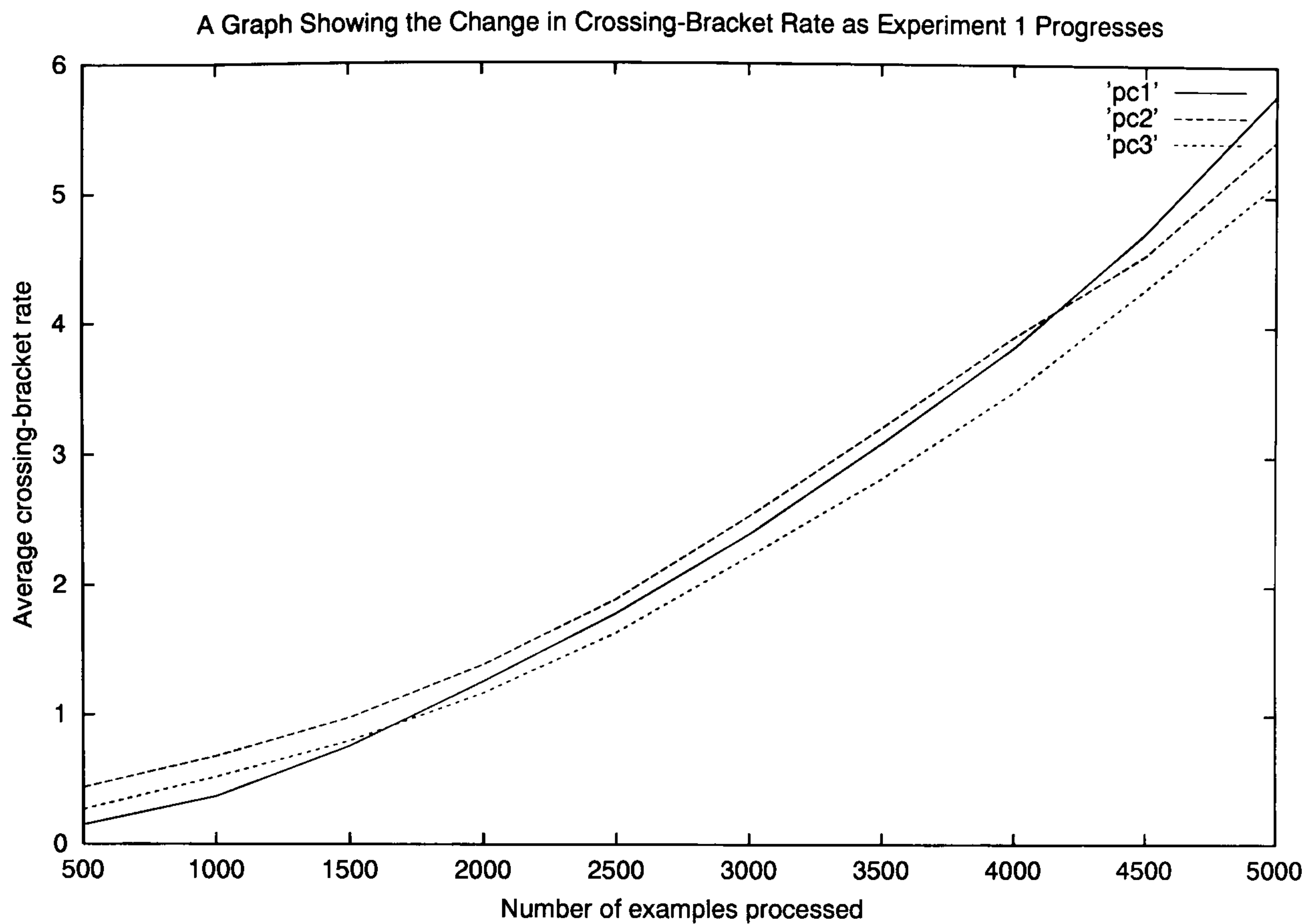


Figure 11.3: A graph showing the change in average crossing-bracket rate for experiment 1

corpus, which are hard to evaluate and give a low accuracy.

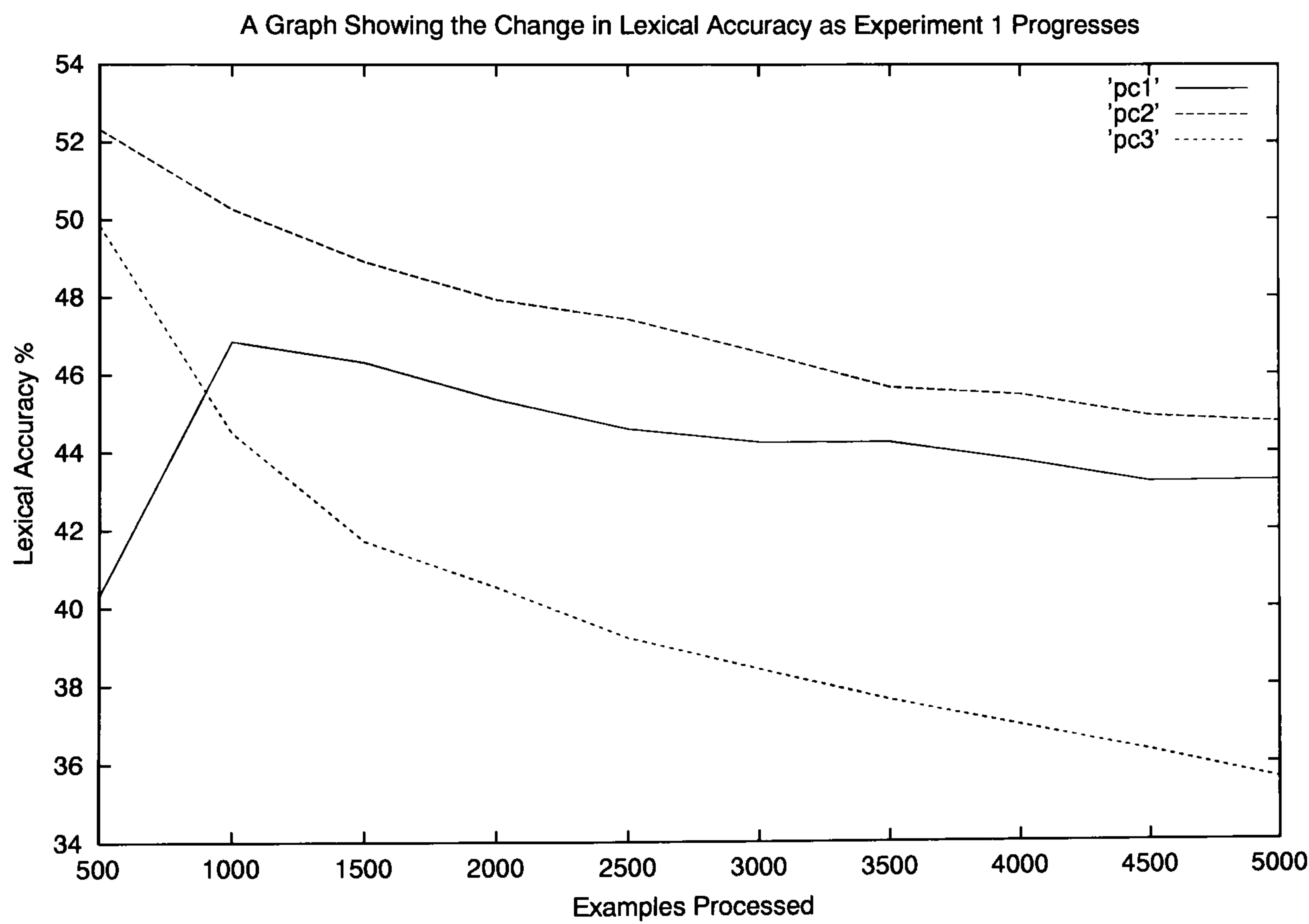


Figure 11.4: A graph showing the change in lexical accuracy for experiment 1

The encouraging aspect of these curves is that the gradient is flattening out and maybe even going up again (the last data point for PC1). Hence it would appear, from a lexical accuracy point of view that, while the increasing lexicon and example size have some impact in decreasing accuracy, the labelling does appear to be close to reaching its minimum accuracy



and that increased data will at worst not cause much degradation in lexical accuracy. It is to be hoped that more data would actually provide an increase in accuracy, as the data became less sparse. This provides further motivation for doing experiments over more data, however. the comments above about the problems of performing such experiments are applicable here too.

The degradation in the lexical accuracy is more pronounced with the experiments with PC3. This suggests that while the NV annotation is sufficient for lexical labelling with simple examples, it does not provide enough of a constraint as the examples get more complex. It should be remembered that this experiment does not include any initial lexicon (i.e. this graph is for the experiment where no bootstrapping lexicon is provided, and the system only has the noun and verb annotation as lexical background knowledge). In future, it may be worth investigating ways of providing information to improve the lexical accuracy when using an NV-annotated corpus.

## 11.7 System Comparisons

In this section the results achieved with CLL are compared against those achieved by some of the other systems presented in Chapter 4. These comparisons are difficult to make, as the systems have been used on different corpora and are often doing rather different tasks. However, it is possible to provide some intuition as to how CLL is performing with respect to these systems.

Firstly, we will consider the CG induction work of Osborne [95] and Osborne and Briscoe [96]. The system is discussed in Chapter 4. The system is trained on 43,000 sentences from the British National Corpus (which has an average sentence length of 17.30 words) and it is tested on 429 sentences from the Spoken English Corpus SEC (which has an average sentence length of 9.94 words)<sup>1</sup>. Assuming that the examples from the SEC were selected at random then the comparison with the shorter corpora from the Penn Treebank is reasonable. They present a system that achieves an average crossing-bracket rate of around 3. This is somewhat better than most of the results presented here (which range from 3.78 to 11.06) and this is especially true, given that the system is tested on spoken English. However, in this context, it should be remembered that the system was significantly more supervised (using part-of-speech sequences as examples, a supervised tri-gram model to evaluate hypotheses and a pre-computed table to indicate head children). Hence the fact that the better results using CLL, which are on corpora of approximately the same length, are close with respect to the crossing-bracket measure is encouraging.

Osborne and Briscoe [96] also give results with respect to coverage (95%) and over-generation (0%). These figures have slightly less meaning in the context of this work. The sentences that can be parsed are predetermined by the categories that the system is allowed (including whether or not closed-class categories can be used for other words). Coverage, depending on the setting, is between 78.5% and 96.3%, based on the sentences in the test set that are parsed. The lexicons developed from the NV-annotated corpora give better coverage. Experiments that use an initial bootstrapping lexicon give better results when

---

<sup>1</sup>For more information on these corpora check: <http://www.hcu.ox.ac.uk/BNC>.



that lexicon is larger. Similarly (and not surprisingly), experiment 5, where the category database is larger, gives greater coverage. However, these results are purely based on the setting rather than the learning process. Given this, results on over-generation have not been calculated.

An interesting alternative comparison is a recent wide-coverage CG (actually a CCG) parser developed by Hockenmaier [66]. Although this is still in the early stages of development and has been applied to a corpus including larger examples and movement (the WSJ section of the Penn Treebank with examples of size forty words or less), it does also have a full grammar present already. In fact, the lexicon that is present could be considered to be the target that CLL is attempting to learn. The average crossing-bracket rate is 4.66, which while higher than many of the results presented here, should probably be compared with results for PC4 (a best of 9.57) as these contain longer examples. While this indicates that CLL has some way to go before learning a good grammar, it also indicates that it will be hard to compare crossing-bracket rates with non-CG systems, as these system may have a much higher goal. The accuracy of the lexical categories is much higher than any of our experiments (perhaps not surprisingly, given that a lexicon is supplied), giving a value of 86.51%. Again, however, it indicates the sort of goal that CLL has. Lexical accuracies are not especially close yet (a best of just over 50% on shorter corpora), however this does mean that instead of being 50% short of a good lexicon, the system is more like 35% short.

A similar comparison can be made with the supertagging work of Joshi and Srinivas [72, 10]. The earlier work produced a best lexical accuracy result using a similar mix of symbolic and statistical constraints as is used in CLL. This gave a lexical accuracy of 77.26%. For these experiments training occurred on part-of-speech tag sequences rather than word sequences and the examples were of a similar length to those in PC1, PC2 and PC3. In supertagging, a full lexicon is supplied, drastically reducing the degree of ambiguity in comparison with CLL. In terms of CLL, if a complete lexicon was available then the ambiguity of it would be around 1.25 categories per word according to the values calculated for the gold standard corpora. In this case, if categories were assigned randomly, then the expected accuracy would be around 80% (this would suggest the ambiguity was higher for the LTAG lexicons used in supertagging). Although it may be a somewhat unfair comparison, as the CG lexicons and the LTAG lexicons may be very different, it would appear that this supertagging model is only learning a fairly small amount in terms of disambiguation. However, in the later work [10], a tri-gram model did achieve over 90%, which is significantly better, although still only a 10% increase on the approximate baseline given above. In this context, however, the increases in accuracy over the random and np baselines that are achieved by CLL are reasonable, although they are still too low to be of much practical use.

Less encouraging comparisons can be made with the work of Clark (CDC) [40], Vervoort (EMILE) [129] and van Zaamen (ABL) [128]. Clark [40] provides results that allow comparison of these unsupervised clustering approaches. They return average crossing-bracket rates from 0.82 to 2.12. The best results from CLL are 1.5 crosses worse than the worst results presented. However, there are some mitigating factors. Firstly, the systems were tested on the ATIS corpus rather than the Penn Treebank. This corpus has little lexical ambiguity [40] and generally consists of short examples over a fairly small domain, so it is



at least comparable with the corpora containing short examples used to test CLL. Secondly, the recall of constituents – 16.8% for the EMILE system of Vervoort [129], 35.6% for the ABL system of van Zaanen [128] and a best of 34.6% for the CDC system of Clark [40] – for all these systems suggests that only a very small amount of the bracketing has been hypothesised (Clark provides an example of this [40]). It is interesting to note from their results that the crossing-bracket rate increases as the recall increases. As Clark explains [40], the crossing-bracket metric tends to suggest these algorithms are better than they actually are. Thirdly, while EMILE and ABL were used on free text, CDC was trained and tested with part-of-speech sequences and hence is somewhat supervised. Finally, these systems (although not all versions of EMILE) concentrate on learning constituents and bracketings, but do not provide labellings of the type provided by CLL – rather there will be random labels for constituents, so no comparison can be made with respect to labelling accuracy. Hence, while these systems perform better on certain measures than CLL, there are reasons that explain this performance.

The results for CLL seem to show that the system performs respectably when compared with other approaches and, while the results may not be a large step forward of themselves, the aim has been to show that reasonable results could be found for this unsupervised setting.

## 11.8 Conclusions

In this chapter a large range of results have been presented. The initial results achieved with CLL on small generated corpora were very encouraging and suggested the continuation of the experiments with larger and more complex corpora.

The experiments with the larger corpora have been encouraging to some extent, in that they have produced results that indicate a significant level of effective learning. They have also allowed some investigation of the possible parameters of CLL. The most important conclusions that can be drawn from these are:

- the compression metric is important and should include frequency information;
- the NV annotation provided a better structural bias;
- the initial lexicon provided a better lexical bias;
- the two biases did not always work well together.

The results with the corpus with longer examples were somewhat disappointing, although in some ways they followed the patterns that had been developing with the other experiments. However, some of the experiments appeared to indicate that there may be a need to use a larger set of data to try and improve the results. This is something that should be attempted in the future, although it will certainly require developing a system that will complete experiments more quickly. There would appear to be further work to be completed in building a more appropriate bias for the learning process.

However, when compared with other systems, CLL appears to be performing comparably and doing so with an unsupervised learning setting.

# Chapter 12

## Conclusions

In this final chapter, I will aim to draw some conclusions about the work that has been presented. I will also seek to evaluate how well the goals of Chapter 1 have been met and whether the suggested hypotheses have been shown to be correct.

Initially I will discuss the strengths and weaknesses of CLL and the model this implementation is based upon (Section 12.1). This will then be followed by a discussion of the questions raised by this work, which were presented in Chapter 1 (Section 12.2). Some of the possibilities for further work will then be discussed in Section 12.3 and some final comments will be given in Section 12.4.

### 12.1 A General Evaluation of the Categorical Lexicon Learner

In this thesis, I have presented the design and implementation of the Categorical Lexicon Learner, CLL, which is used to build Categorical Grammar (CG) lexicons and in the process annotate corpora of natural language with CG parses.

CLL has both unsupervised and weakly supervised settings, which are intended to allow it to be used wherever there is a large amount of computer-readable text available for the domain for which a lexicon is required. To support this, CLL, only requires positive examples, i.e. correct text examples.

The system is essentially symbolic, i.e. the input and the output are symbolic and the learning process involves manipulating the symbolic input. However, some simple stochastic methods have been added to the symbolic methods so that both the Maximum-Likelihood (ML) principle and a compression principle (which is somewhat MDL-like) have been used together to heuristically guide the learning process.

The stated aim in Chapter 1 was to build a useful system, by which it was meant that the problem solved would be useful (i.e. it is a real problem) and that the output of the system as it attempts to solve the problem would also be useful. CLL is designed to solve a useful problem. Building grammars for domains without the need either for a lot of manual grammar building, or manual construction of large resources (e.g. annotated corpora, or lexicons) can allow the faster construction of NLP systems for a very wide range of domains. It is the case that some domain-specific resources are still required by CLL (the category database and the weak annotation or the initial lexicon), however, these are considered to



be relatively easy to build. In future, it may be desirable to reduce the cost of building these resources by improving the learning methods in CLL.

The output of CLL is also potentially useful. If the lexicons built by CLL were of high quality then it would be possible to start using the system to build natural language resources. However, the current results do not indicate this kind of quality of output. CLL is perhaps best seen, therefore, as a first step towards building both large-scale CG lexicons and CG annotated corpora. That is not to say that the output from CLL is currently useless. Crossing-bracket rates of around 4 and lexical accuracies of around 50% (albeit on relatively small corpora) indicate a significant step forward and the lexicons and corpora generated by CLL could perhaps be used as a basis either for accurate hand-built lexicons and corpora, or for some more supervised learning methods. It should be noted that these results also compare reasonably favourably with other systems, which attempt to perform similar learning tasks.

While the above discussion indicates that CLL has to a large extent met the aims set out in Chapter 1, there are also a number of weaknesses that need to be mentioned. These weaknesses are mostly centred on the issue of efficiency. The repeated use of full parsing causes the system to lack efficiency. This in turn means that less data can be dealt with in a reasonable time, which has led to sparse data problems being significant. Furthermore, the current system is limited in its syntactic coverage (e.g. movement is not currently dealt with) and any attempt to improve the coverage will lead to further efficiency problems with respect to the parsing. Future work will need to concentrate on ways of improving the efficiency, as this in itself, by allowing more data to be processed, may cause an improvement in the results.

Some problems have also been identified in the use of the stochastic methods. It would appear that the uni-gram based stochastic Categorical Grammar is not a good enough stochastic model and that more context needs to be taken into account by the probability model. Along with this, it would appear that the compression metric and ML principle do not necessarily combine very well in learning, so further investigation of learning principles will be important.

Despite these weaknesses, CLL, has, to a large extent, met the aims set out in Chapter 1 and has been shown to perform a significant degree of learning.

## 12.2 Evaluating the General Contributions

In Chapter 1, a number of questions were presented that the work in this thesis has, to some extent, provided answers to. In this section the answers or partial answers that have been suggested by the work presented here are summarised and the overall contributions of the work, are also presented.

Firstly, CLL appears to show that it *is* possible to build unsupervised (or weakly supervised), positive-only, natural-language syntax learning systems. However, there are two caveats to this. Firstly, the results show only a limited degree of success in learning useful syntax. Secondly, to achieve even this success, the system requires a large amount of background knowledge.

Secondly, there is the issue of whether or not simple stochastic methods, applied with symbolic methods, are useful. Work with CLL seems to indicate that, while simple stochastic methods do allow a significant amount to be achieved (the degree of learning indicated in Chapter 11 shows this), it would also appear that both more complicated stochastic models of language (in particular taking into account context) and more effective learning principles may allow further improvement in performance.

Thirdly, in building CLL, the type of background knowledge that is useful has been addressed. The background knowledge provided to CLL can include:

- an initial lexicon,
- a complete set of language-specific categories,
- a complete set of category combination rules.

In practice this knowledge is reasonably easy to construct, especially when compared with the effort needed to construct annotated data or negative examples. However, it is also the case that it would be preferable, both from a practical perspective and a psychological perspective, to remove the language-specific parts of the background knowledge.

The evidence with respect to the use of the maximum-likelihood principle and compression principle is again somewhat mixed. The results show that these two principles are guiding the system towards a significant degree of learning. However, it does also appear from the results that the two principles clash to some extent (for example eliminating the effect of compression by only calculating the best parse often produces some of the best results). In future further investigation of types of learning principles and their interaction should be pursued.

The final question investigated was based upon the suitability of a lexicalised formalism, like CG, for learning tasks. It would appear that this formalism is very suitable, as it allows the task to be clearly defined (reducing the problem from learning a lexicon and a set of grammar rules, to just learning a lexicon). In practice, this allows the use of more focussed methods of learning. It would appear that the increasing preference for lexicalised grammars in the Computational Linguistics community should be followed in the Natural Language Learning community.

The major contributions of this work are therefore:

- an unsupervised, positive-only learning system for CG lexicons,
- a weakly supervised, positive-only learning system for CG lexicons,
- an investigation of how effectively these systems work and which parameter settings return the best results for a set of evaluation metrics.

In general, the final of these contributions can be summarised in the following way. To maximise the parse accuracy of the lexicon, the weakly supervised system is preferable, using the most MDL-like compression, a small initial lexicon and quite a small beam for the parser. To maximise the lexical accuracy, the unsupervised setting is much better. In particular this setting should be used with a large initial lexicon and a relatively small beam. In general



smaller beams gave better results and the inclusion of the NP rule consistently gave better performance.

Some other useful work has also been presented, in particular the construction of a system for the translation of corpus annotation, which has been used to give a preliminary CG annotation for the Penn Treebank. In principle, such a system could be very useful for NLP researchers who wish to work in formalisms for which there are very few resources. This work could be profitably pursued with this in mind.

## 12.3 Further Work

The discussion above indicates that there are many opportunities for further work in this area. However, in this section, I will limit the discussion to three areas.

- developing CLL
- experimenting with CLL
- investigating better evaluation techniques

### 12.3.1 Developing the System

In this section, the extensions to CLL that are being considered are brought together and discussed with respect to future work. Most of these developments come initially from a desire to improve either the performance of the system or the psychological plausibility of the system.

Perhaps the first modification of the system would be to extend the grammatical background knowledge so that the parser can deal effectively with movement. Such a modification would allow the system to be used on full natural-language corpora. To achieve this coverage the system would need to use the extra expressive power of a formalism like Combinatory Categorical Grammar (CCG) [118, 119], which would also allow more elegant handling of other syntactic structures such as a coordination.

However, to achieve this a number of efficiency improvements would need to be implemented, as the CCG formalism would at least require more rules and probably more categories, which would have a large impact on the efficiency of the parser. In particular, the current reparsing approach would have to be made more efficient and it would probably be necessary to add further constraints to the parser.

An approach that no longer reparsed previous examples would therefore be advantageous both from an efficiency perspective and from the perspective of psychological plausibility. One possibility would be using a thresholding method with lexical entries, such that once their frequency drops below a certain threshold, the entries are removed and examples using those entries are reparsed (if an accurate parse annotation is required), however some further work would perhaps be necessary to ensure that the learner continued to build linguistically plausible lexicons.

It would also be interesting to use a system that could build partial analyses and thus deal more effectively with noisy data, or for example spoken rather than written language. In this context, the use of a robust partial parser might be interesting.

One of the most interesting developments would be the use of other compression metrics. In particular, it would be interesting to apply models such as Minimum Description Length (MDL) learning. Given the fact that the compression metric can make a sizeable impact, this could be a particularly fruitful area of further work.

A stronger stochastic model of language that included context could also be developed, e.g. some kind of  $n$ -gram model where  $n > 1$ . Given the effectiveness of such models and the psychological plausibility of using probabilistic methods, this would seem an ideal way to strengthen the constraints on the learning system.

Finally, the related issue of defining a better set of constraints or bias for the learning algorithm needs to be addressed. It appears that the current settings, while useful, are not powerful enough, especially as the example length grows.

### 12.3.2 Further Experimentation

There are clearly a great number more experiments that could be performed using CLL. A simple next step would be to perform further experiments with PC4 and experiment with part-of-speech tagged corpora to allow for better evaluation of the current performance of CLL on these corpora. It would also be interesting to build the resources to allow evaluation of the current system on a corpus from the Penn Treebank including examples with movement.

It may also be valuable to do a few more directed experiments to see how other parameters interact. For example, the corpus-compression metric appears to cause a reasonable improvement on the whole. Further experiments could be performed with other parameters that improved performance (e.g. a large initial lexicon in some cases) to attempt to achieve the best possible results with the current version of CLL.

Obviously it is also the case that all of the extensions of the system discussed above would lead to a large number of further experiments to be performed.

### 12.3.3 Evaluation

Finally, it is not easy to evaluate unsupervised syntax-learning systems. Firstly, there are commonly not the resources available for testing (e.g. CG annotated corpora). Secondly, the metrics commonly used give a somewhat abstract notion of the quality of the syntactic knowledge learned, as they are really parsing metrics.

These problems lead to two suggestions for further work. Firstly, it is necessary to work on building better resources. In particular it would be interesting to extend the work reported in Chapter 10 on translating corpora into different formalisms. This process could be made more effective for CG and could possibly be extended to translate corpora into other formalisms.

Secondly, further work needs to be put into considering more appropriate methods and metrics for evaluating natural-language syntax-learning systems. For example, further work could involve investigating sensible baseline results.



## 12.4 Conclusions

The design and implementation of the Categorical Lexicon Learner, CLL, has both provided the first step to building a useful system that can learn Categorical Grammar lexicons from unannotated, positive only examples. The results presented are encouraging given the difficulty of the problem being considered. The work completed has opened up a large number of areas for future work, especially in the areas of improving coverage, improving the statistical models and investigating further learning principles.

Hence, this thesis provides useful material for those who are interested using in Machine Learning in Natural Language Processing, especially those who are interested learning syntax, or more specifically Categorical Grammar.

## Appendix A

# The Penn Part-of-Speech Annotation

In this appendix I include a description of all the part-of-speech tags in the version of the Penn Treebank that has been used during the work described in this thesis. The information is extracted directly from Santorini’s tagging guidelines [110] and the documentation with the treebank itself.

CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential “there”
FW	Foreign word
IN	Preposition or suboordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP\$	Possesive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	“to”



UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VCN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP\$	Possessive wh-pronoun
WRB	Wh-adverb
\$	dollar sign, also US\$, NZ\$ eyc.
#	Pound sign (usually sterling)
“	Left double or single quote
”	Right double or single quote
(	Left parenthesis (any bracket type)
)	Right parenthesis (any bracket type)
,	Comma
.	End of sentence punctuation (., !, ?)
:	Mid-sentence punctuation (:, ;, ..., -)

# Appendix B

## Initial Lexicons

The initial lexicons used with the experiments are presented in this section. They are presented in the form that they are used with CLL.

### B.1 CCW1

```
%%% Name: lex.pl
%%% Author: S.P. Watkinson
%%% Date Started: 8/2/99
%%% Date Last Modified: 8/2/99
%%% Purpose: A partial probabilistic lexicon containing closed
%%%          class words.
```

```
:- dynamic lex/3.
```

```
% Prepositions
```

```
lex(cc(on), fs(bs(n, n), np), 0.5).
lex(cc(on), fs(bs(bs(s, np), bs(s, np)), np), 0.5).
lex(cc(with), fs(bs(n, n), np), 0.5).
lex(cc(with), fs(bs(bs(s, np), bs(s, np)), np), 0.5).
lex(cc(in), fs(bs(n, n), np), 0.5).
lex(cc(in), fs(bs(bs(s, np), bs(s, np)), np), 0.5).
lex(cc(inside), fs(bs(n, n), np), 0.5).
lex(cc(inside), fs(bs(bs(s, np), bs(s, np)), np), 0.5).
lex(cc(under), fs(bs(bs(s, np), bs(s, np)), np), 0.5).
lex(cc(under), fs(bs(n, n), np), 0.5).
lex(cc(beside), fs(bs(bs(s, np), bs(s, np)), np), 0.5).
lex(cc(beside), fs(bs(n, n), np), 0.5).
lex(cc(from), fs(bs(bs(s, np), bs(s, np)), np), 0.5).
lex(cc(from), fs(bs(n, n), np), 0.5).
lex(cc(above), fs(bs(bs(s, np), bs(s, np)), np), 0.5).
```



```
lex(cc(above), fs(bs(n, n), np), 0.5).
lex(cc(at), fs(bs(bs(s, np), bs(s, np)), np), 0.5).
lex(cc(at), fs(bs(n, n), np), 0.5).
lex(cc(to), fs(bs(bs(s, np), bs(s, np)), np), 0.5).
lex(cc(to), fs(bs(n, n), np), 0.5).
```

#### % Determiners

```
lex(cc(a), fs(np, n), 1).
lex(cc(the), fs(np, n), 1).
lex(cc(some), fs(np, n), 1).
lex(cc(no), fs(np, n), 1).
lex(cc(every), fs(np, n), 1).
lex(cc(which), fs(np, n), 1).
lex(cc(what), fs(np, n), 1).
lex(cc(all), fs(np, n), 1).
```

#### % Conjunctions

```
lex(cc(and), bs(fs(s, s), s), 1).
lex(cc(or), bs(fs(s, s), s), 1).
lex(cc(but), bs(fs(s, s), s), 1).
```

## B.2 CCW2

```
%%% Name: lex.pl
%%% Author: S.P. Watkinson
%%% Date Started: 8/2/99
%%% Date Last Modified: 3/10/00
%%% Purpose: A partial probabilistic lexicon containing closed
%%%          class words.
```

```
:- dynamic lex/3.
```

#### % Prepositions

```
lex(cc(to), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(to), fs(bs(n, n), np), 0.35).
lex(cc(to), fs(fs(s, s), np), 0.3).
lex(cc(in), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(in), fs(bs(n, n), np), 0.35).
```

```

lex(cc(in), fs(fs(s, s), np), 0.3).
lex(cc(of), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(of), fs(bs(n, n), np), 0.35).
lex(cc(of), fs(fs(s, s), np), 0.3).
lex(cc(on), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(on), fs(bs(n, n), np), 0.35).
lex(cc(on), fs(fs(s, s), np), 0.3).
lex(cc(about), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(about), fs(bs(n, n), np), 0.35).
lex(cc(about), fs(fs(s, s), np), 0.3).
lex(cc(before), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(before), fs(bs(n, n), np), 0.35).
lex(cc(before), fs(fs(s, s), np), 0.3).
lex(cc(as), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(as), fs(bs(n, n), np), 0.35).
lex(cc(as), fs(fs(s, s), np), 0.3).
lex(cc(across), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(across), fs(bs(n, n), np), 0.35).
lex(cc(across), fs(fs(s, s), np), 0.3).
lex(cc(for), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(for), fs(bs(n, n), np), 0.35).
lex(cc(for), fs(fs(s, s), np), 0.3).
lex(cc(from), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(from), fs(bs(n, n), np), 0.35).
lex(cc(from), fs(fs(s, s), np), 0.3).
lex(cc(below), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(below), fs(bs(n, n), np), 0.35).
lex(cc(below), fs(fs(s, s), np), 0.3).
lex(cc(since), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(since), fs(bs(n, n), np), 0.35).
lex(cc(since), fs(fs(s, s), np), 0.3).
lex(cc(like), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(like), fs(bs(n, n), np), 0.35).
lex(cc(like), fs(fs(s, s), np), 0.3).
lex(cc(than), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(than), fs(bs(n, n), np), 0.35).
lex(cc(than), fs(fs(s, s), np), 0.3).
lex(cc(that), fs(bs(bs(s, np), bs(s, np)), np), 0.25).
lex(cc(that), fs(bs(n, n), np), 0.25).
lex(cc(that), fs(fs(s, s), np), 0.25).
lex(cc(with), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(with), fs(bs(n, n), np), 0.35).
lex(cc(with), fs(fs(s, s), np), 0.3).

```



```

lex(cc(by), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(by), fs(bs(n, n), np), 0.35).
lex(cc(by), fs(fs(s, s), np), 0.3).
lex(cc(at), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(at), fs(bs(n, n), np), 0.35).
lex(cc(at), fs(fs(s, s), np), 0.3).
lex(cc(off), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(off), fs(bs(n, n), np), 0.35).
lex(cc(off), fs(fs(s, s), np), 0.3).
lex(cc(ago), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(ago), fs(bs(n, n), np), 0.35).
lex(cc(ago), fs(fs(s, s), np), 0.3).
lex(cc(through), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(through), fs(bs(n, n), np), 0.35).
lex(cc(through), fs(fs(s, s), np), 0.3).
lex(cc(between), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(between), fs(bs(n, n), np), 0.35).
lex(cc(between), fs(fs(s, s), np), 0.3).
lex(cc(until), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(until), fs(bs(n, n), np), 0.35).
lex(cc(until), fs(fs(s, s), np), 0.3).
lex(cc(behind), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(behind), fs(bs(n, n), np), 0.35).
lex(cc(behind), fs(fs(s, s), np), 0.3).
lex(cc(up), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(up), fs(bs(n, n), np), 0.35).
lex(cc(up), fs(fs(s, s), np), 0.3).
lex(cc('@'), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc('@'), fs(bs(n, n), np), 0.35).
lex(cc('@'), fs(fs(s, s), np), 0.3).
lex(cc(so), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(so), fs(bs(n, n), np), 0.35).
lex(cc(so), fs(fs(s, s), np), 0.3).
lex(cc(under), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(under), fs(bs(n, n), np), 0.35).
lex(cc(under), fs(fs(s, s), np), 0.3).
lex(cc(unlike), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(unlike), fs(bs(n, n), np), 0.35).
lex(cc(unlike), fs(fs(s, s), np), 0.3).
lex(cc(against), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(against), fs(bs(n, n), np), 0.35).
lex(cc(against), fs(fs(s, s), np), 0.3).
lex(cc(over), fs(bs(bs(s, np), bs(s, np)), np), 0.35).

```

```

lex(cc(over), fs(bs(n, n), np), 0.35).
lex(cc(over), fs(fs(s, s), np), 0.3).
lex(cc(if), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(if), fs(bs(n, n), np), 0.35).
lex(cc(if), fs(fs(s, s), np), 0.3).
lex(cc(per), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(per), fs(bs(n, n), np), 0.35).
lex(cc(per), fs(fs(s, s), np), 0.3).
lex(cc(after), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(after), fs(bs(n, n), np), 0.35).
lex(cc(after), fs(fs(s, s), np), 0.3).
lex(cc(while), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(while), fs(bs(n, n), np), 0.35).
lex(cc(while), fs(fs(s, s), np), 0.3).
lex(cc(around), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(around), fs(bs(n, n), np), 0.35).
lex(cc(around), fs(fs(s, s), np), 0.3).
lex(cc(whether), fs(bs(bs(s, np), bs(s, np)), np), 0.16).
lex(cc(whether), fs(bs(n, n), np), 0.16).
lex(cc(whether), fs(fs(s, s), np), 0.16).
lex(cc(because), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(because), fs(bs(n, n), np), 0.35).
lex(cc(because), fs(fs(s, s), np), 0.3).
lex(cc(down), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(down), fs(bs(n, n), np), 0.35).
lex(cc(down), fs(fs(s, s), np), 0.3).
lex(cc(out), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(out), fs(bs(n, n), np), 0.35).
lex(cc(out), fs(fs(s, s), np), 0.3).
lex(cc(among), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(among), fs(bs(n, n), np), 0.35).
lex(cc(among), fs(fs(s, s), np), 0.3).
lex(cc(though), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(though), fs(bs(n, n), np), 0.35).
lex(cc(though), fs(fs(s, s), np), 0.3).
lex(cc(outside), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(outside), fs(bs(n, n), np), 0.35).
lex(cc(outside), fs(fs(s, s), np), 0.3).
lex(cc(during), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(during), fs(bs(n, n), np), 0.35).
lex(cc(during), fs(fs(s, s), np), 0.3).
lex(cc('As'), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc('As'), fs(bs(n, n), np), 0.35).

```



```

lex(cc('As'), fs(fs(s, s), np), 0.3).
lex(cc('In'), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc('In'), fs(bs(n, n), np), 0.35).
lex(cc('In'), fs(fs(s, s), np), 0.3).
lex(cc(without), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(without), fs(bs(n, n), np), 0.35).
lex(cc(without), fs(fs(s, s), np), 0.3).
lex(cc(except), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(except), fs(bs(n, n), np), 0.35).
lex(cc(except), fs(fs(s, s), np), 0.3).
lex(cc(despite), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(despite), fs(bs(n, n), np), 0.35).
lex(cc(despite), fs(fs(s, s), np), 0.3).
lex(cc(into), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(into), fs(bs(n, n), np), 0.35).
lex(cc(into), fs(fs(s, s), np), 0.3).
lex(cc(via), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(via), fs(bs(n, n), np), 0.35).
lex(cc(via), fs(fs(s, s), np), 0.3).
lex(cc(toward), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(toward), fs(bs(n, n), np), 0.35).
lex(cc(toward), fs(fs(s, s), np), 0.3).
lex(cc(next), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(next), fs(bs(n, n), np), 0.35).
lex(cc(next), fs(fs(s, s), np), 0.3).
lex(cc('OF'), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc('OF'), fs(bs(n, n), np), 0.35).
lex(cc('OF'), fs(fs(s, s), np), 0.3).
lex(cc(near), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(near), fs(bs(n, n), np), 0.35).
lex(cc(near), fs(fs(s, s), np), 0.3).
lex(cc(unless), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(unless), fs(bs(n, n), np), 0.35).
lex(cc(unless), fs(fs(s, s), np), 0.3).
lex(cc(above), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(above), fs(bs(n, n), np), 0.35).
lex(cc(above), fs(fs(s, s), np), 0.3).
lex(cc(although), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(although), fs(bs(n, n), np), 0.35).
lex(cc(although), fs(fs(s, s), np), 0.3).
lex(cc('For'), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc('For'), fs(bs(n, n), np), 0.35).
lex(cc('For'), fs(fs(s, s), np), 0.3).

```

```

lex(cc(throughout), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(throughout), fs(bs(n, n), np), 0.35).
lex(cc(throughout), fs(fs(s, s), np), 0.3).
lex(cc('That'), fs(bs(bs(s, np), bs(s, np)), np), 0.25).
lex(cc('That'), fs(bs(n, n), np), 0.25).
lex(cc('That'), fs(fs(s, s), np), 0.25).
lex(cc('v.'), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc('v.'), fs(bs(n, n), np), 0.35).
lex(cc('v.'), fs(fs(s, s), np), 0.3).
lex(cc(along), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(along), fs(bs(n, n), np), 0.35).
lex(cc(along), fs(fs(s, s), np), 0.3).
lex(cc(within), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(within), fs(bs(n, n), np), 0.35).
lex(cc(within), fs(fs(s, s), np), 0.3).
lex(cc('Of'), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc('Of'), fs(bs(n, n), np), 0.35).
lex(cc('Of'), fs(fs(s, s), np), 0.3).
lex(cc(onto), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(onto), fs(bs(n, n), np), 0.35).
lex(cc(onto), fs(fs(s, s), np), 0.3).
lex(cc(beneath), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(beneath), fs(bs(n, n), np), 0.35).
lex(cc(beneath), fs(fs(s, s), np), 0.3).
lex(cc(unto), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(unto), fs(bs(n, n), np), 0.35).
lex(cc(unto), fs(fs(s, s), np), 0.3).
lex(cc('Because'), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc('Because'), fs(bs(n, n), np), 0.35).
lex(cc('Because'), fs(fs(s, s), np), 0.3).
lex(cc(inside), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(inside), fs(bs(n, n), np), 0.35).
lex(cc(inside), fs(fs(s, s), np), 0.3).
lex(cc(beside), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(beside), fs(bs(n, n), np), 0.35).
lex(cc(beside), fs(fs(s, s), np), 0.3).
lex(cc(upon), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(upon), fs(bs(n, n), np), 0.35).
lex(cc(upon), fs(fs(s, s), np), 0.3).
lex(cc(de), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(de), fs(bs(n, n), np), 0.35).
lex(cc(de), fs(fs(s, s), np), 0.3).
lex(cc(once), fs(bs(bs(s, np), bs(s, np)), np), 0.35).

```



```

lex(cc(once), fs(bs(n, n), np), 0.35).
lex(cc(once), fs(fs(s, s), np), 0.3).
lex(cc(besides), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(besides), fs(bs(n, n), np), 0.35).
lex(cc(besides), fs(fs(s, s), np), 0.3).
lex(cc('vs.'), fs(bs(bs(s, np), bs(s, np)), np), 0.16).
lex(cc('vs.'), fs(bs(n, n), np), 0.16).
lex(cc('vs.'), fs(fs(s, s), np), 0.16).
lex(cc(beyond), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(beyond), fs(bs(n, n), np), 0.35).
lex(cc(beyond), fs(fs(s, s), np), 0.3).
lex(cc(but), fs(bs(bs(s, np), bs(s, np)), np), 0.16).
lex(cc(but), fs(bs(n, n), np), 0.16).
lex(cc(but), fs(fs(s, s), np), 0.16).
lex(cc(notwithstanding), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(notwithstanding), fs(bs(n, n), np), 0.35).
lex(cc(notwithstanding), fs(fs(s, s), np), 0.3).
lex(cc(past), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(past), fs(bs(n, n), np), 0.35).
lex(cc(past), fs(fs(s, s), np), 0.3).
lex(cc(amid), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(amid), fs(bs(n, n), np), 0.35).
lex(cc(amid), fs(fs(s, s), np), 0.3).
lex(cc('Whether'), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc('Whether'), fs(bs(n, n), np), 0.35).
lex(cc('Whether'), fs(fs(s, s), np), 0.3).
lex(cc(iF), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(iF), fs(bs(n, n), np), 0.35).
lex(cc(iF), fs(fs(s, s), np), 0.3).
lex(cc('IN'), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc('IN'), fs(bs(n, n), np), 0.35).
lex(cc('IN'), fs(fs(s, s), np), 0.3).
lex(cc(astride), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(astride), fs(bs(n, n), np), 0.35).
lex(cc(astride), fs(fs(s, s), np), 0.3).
lex(cc(aboard), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(aboard), fs(bs(n, n), np), 0.35).
lex(cc(aboard), fs(fs(s, s), np), 0.3).
lex(cc(lest), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(lest), fs(bs(n, n), np), 0.35).
lex(cc(lest), fs(fs(s, s), np), 0.3).
lex(cc('With'), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc('With'), fs(bs(n, n), np), 0.35).

```

```

lex(cc('With'), fs(fs(s, s), np), 0.3).
lex(cc(atop), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(atop), fs(bs(n, n), np), 0.35).
lex(cc(atop), fs(fs(s, s), np), 0.3).
lex(cc(plus), fs(bs(bs(s, np), bs(s, np)), np), 0.16).
lex(cc(plus), fs(bs(n, n), np), 0.16).
lex(cc(plus), fs(fs(s, s), np), 0.16).
lex(cc(aMONG), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(aMONG), fs(bs(n, n), np), 0.35).
lex(cc(aMONG), fs(fs(s, s), np), 0.3).
lex(cc(tHROUGHOUT), fs(bs(bs(s, np), bs(s, np)), np), 0.35).
lex(cc(tHROUGHOUT), fs(bs(n, n), np), 0.35).
lex(cc(tHROUGHOUT), fs(fs(s, s), np), 0.3).

```

#### % Determiners

```

lex(cc(a), fs(np, n), 1).
lex(cc(the), fs(np, n), 1).
lex(cc(some), fs(np, n), 1).
lex(cc(no), fs(np, n), 1).
lex(cc(every), fs(np, n), 1).
lex(cc(which), fs(np, n), 1).
lex(cc(what), fs(np, n), 1).
lex(cc(all), fs(np, n), 1).
lex(cc(this), fs(np, n), 1).
lex(cc(those), fs(np, n), 1).
lex(cc(these), fs(np, n), 1).
lex(cc(an), fs(np, n), 1).
lex(cc(any), fs(np, n), 1).
lex(cc(both), fs(np, n), 0.25).
lex(cc(another), fs(np, n), 1).
lex(cc(that), fs(np, n), 0.25).
lex(cc(half), fs(np, n), 1).
lex(cc('This'), fs(np, n), 1).
lex(cc(each), fs(np, n), 1).
lex(cc('The'), fs(np, n), 1).
lex(cc('Both'), fs(np, n), 1).
lex(cc(either), fs(np, n), 1).
lex(cc('All'), fs(np, n), 1).
lex(cc('tHE'), fs(np, n), 1).
lex(cc('A'), fs(np, n), 1).
lex(cc(neither), fs(np, n), 1).
lex(cc(bOTH), fs(np, n), 1).

```



```
lex(cc('That'), fs(np, n), 0.25).
lex(cc(many), fs(np, n), 1).
lex(cc('THE'), fs(np, n), 1).
```

#### % Conjunctions

```
lex(cc(and), bs(fs(s, s), s), 0.33).
lex(cc(and), bs(fs(np, np), np), 0.33).
lex(cc(and), fs(s, s), 0.33).
lex(cc('And'), bs(fs(s, s), s), 0.33).
lex(cc('And'), bs(fs(np, np), np), 0.33).
lex(cc('And'), fs(s, s), 0.33).
lex(cc('AND'), bs(fs(s, s), s), 0.33).
lex(cc('AND'), bs(fs(np, np), np), 0.33).
lex(cc('AND'), fs(s, s), 0.33).
lex(cc(or), bs(fs(s, s), s), 0.33).
lex(cc(or), bs(fs(np, np), np), 0.33).
lex(cc(or), fs(s, s), 0.33).
lex(cc(but), bs(fs(s, s), s), 0.16).
lex(cc(but), bs(fs(np, np), np), 0.16).
lex(cc(but), fs(s, s), 0.16).
lex(cc('vs.'), bs(fs(s, s), s), 0.16).
lex(cc('vs.'), bs(fs(np, np), np), 0.16).
lex(cc('vs.'), fs(s, s), 0.16).
lex(cc('&'), bs(fs(s, s), s), 0.33).
lex(cc('&'), bs(fs(np, np), np), 0.33).
lex(cc('&'), fs(s, s), 0.33).
lex(cc(either), bs(fs(s, s), s), 0.33).
lex(cc(either), bs(fs(np, np), np), 0.33).
lex(cc(either), fs(s, s), 0.33).
lex(cc(nor), bs(fs(s, s), s), 0.33).
lex(cc(nor), bs(fs(np, np), np), 0.33).
lex(cc(nor), fs(s, s), 0.33).
lex(cc(plus), bs(fs(s, s), s), 0.16).
lex(cc(plus), bs(fs(np, np), np), 0.16).
lex(cc(plus), fs(s, s), 0.16).
lex(cc(neither), bs(fs(s, s), s), 0.33).
lex(cc(neither), bs(fs(np, np), np), 0.33).
lex(cc(neither), fs(s, s), 0.33).
lex(cc(both), bs(fs(s, s), s), 0.25).
lex(cc(both), bs(fs(np, np), np), 0.25).
lex(cc(both), fs(s, s), 0.25).
lex(cc(yet), bs(fs(s, s), s), 0.33).
```

```
lex(cc(yet), bs(fs(np, np), np), 0.33).  
lex(cc(yet), fs(s, s), 0.33).  
lex(cc(whether), bs(fs(s, s), s), 0.16).  
lex(cc(whether), bs(fs(np, np), np), 0.16).  
lex(cc(whether), fs(s, s), 0.16).
```



# Appendix C

## The Category Databases

The category databases that have been used with CLL are presented in this appendix as they would be presented to CLL. CATDB1 is simply the set of categories without the extra information, as this was used with an earlier version of CLL.

### C.1 CATDB1

cat(s).	% S
cat(n).	% N
cat(np).	% NP
cat(bs(s, np)).	% IVP
cat(fs(bs(s, np), np)).	% TVP
cat(fs(fs(bs(s, np), np), np)).	% DVP
cat(fs(bs(s, np), s)).	% Sentential comp
cat(fs(np, n)).	% Det
cat(fs(n, n)).	% Adj
cat(fs(bs(s, np), bs(s, np))).	% Aux
cat(fs(bs(n, n), np)).	% NP PP
cat(fs(bs(bs(s, np), bs(s, np)), np)).	% VP PP

### C.2 CATDB2

```
%%% Name: catdb.pl
%%% Author: S.P. Watkinson
%%% Date Started: 23/11/98
%%% Date Last Modified: 17/8/00
%%% Purpose: A database of the Categorical Grammar categories available.
%%%          A module for cll.pl.
%%%
%%% Modifications
%%%
%%% Categories signifcantly increased.  More flexible
```

%%% preposition, adverbs, adjectives. Conjunction added.

%%% Name: cat/1

%%% Arguments: 1. a categorial grammar category (Steedman notation),

%%%                bs/2 is a backslash, fs/2 is a forward slash.

%%% Purpose: Contains a set of facts to denote what categories are

%%%                available.

```
%cat(s).                % S
cat(n).                 % N
cat(np).               % NP
cat(bs(s, np)).        % IVP
cat(fs(bs(s, np), np)). % TVP
cat(fs(fs(bs(s, np), np), np)). % DVP
cat(fs(bs(s, np), s)). % Sentential comp
cat(fs(np, n)).        % Det
cat(fs(n, n)).         % Adj
cat(fs(bs(s, np), bs(s, np))). % Aux
cat(fs(np, s)).        % that
cat(fs(bs(n, n), np)). % NP PP
cat(fs(bs(bs(s, np), bs(s, np)), np)). % VP PP
cat(fs(fs(s, s), np)). % S PP
cat(fs(s, s)).         % S Adv
cat(bs(s, s)).         % S Adv
cat(fs(bs(s, np), bs(s, np))). % V Adv
cat(bs(bs(s, np), bs(s, np))). % V Adv
cat(fs(fs(s, s), fs(s, s))). % Adv Inten
cat(fs(bs(s, s), bs(s, s))). % Adv Inten
cat(fs(fs(bs(s, np), bs(s, np)), fs(bs(s, np), bs(s, np)))). % Adv Inten
cat(fs(bs(bs(s, np), bs(s, np)), bs(bs(s, np), bs(s, np)))). % Adv Inten
cat(fs(fs(bs(n, n), np), fs(bs(n, n), np))). % Adv
cat(bs(fs(bs(n, n), np), fs(bs(n, n), np))). % Adv
cat(fs(fs(bs(bs(s, np), bs(s, np)), np), fs(bs(bs(s, np), bs(s, np)), np))).
%Adv
cat(bs(fs(bs(bs(s, np), bs(s, np)), np), fs(bs(bs(s, np), bs(s, np)), np))).
%Adv
cat(fs(fs(n, n), fs(n, n))). % Adv
%cat(fs(bs(sinf, np), bs(s, np))). % Infinitival to
%cat(fs(bs(s, np), bs(sinf, np))). % Infinitival complement
cat(fs(bs(s, np), fs(n, n))). % Copuler V
cat(fs(bs(s, s), s)). % Sentential CC
cat(fs(bs(np, np), np)). % NP CC
cat(fs(np, np)). % Adj
```



%%% Predicate Name: cat\_num/1.  
 %%% Arguments: Number of categories  
 %%% Purpose: Used to hold the number of categories.

cat\_num(30).

### C.3 CATDB3

cat\_num(45).  
 cat(bs(s,s)).  
 cat(fs(bs(np,np),fs(np,n))).  
 cat(fs(bs(fs(np,np),fs(np,np)),np)).  
 cat(fs(bs(n,n),n)).  
 cat(fs(fs(s,s),fs(s,s))).  
 cat(fs(bs(np,np),s)).  
 cat(bs(bs(np,np),bs(np,np))).  
 cat(fs(bs(bs(s,np),bs(s,np)),fs(bs(np,np),np))).  
 cat(bs(fs(bs(s,np),bs(s,np)),fs(bs(s,np),bs(s,np)))).  
 cat(fs(bs(fs(np,n),fs(np,n)),fs(np,n))).  
 cat(fs(bs(bs(np,np),bs(np,np)),np)).  
 cat(bs(bs(bs(s,np),bs(s,np)),bs(bs(s,np),bs(s,np)))).  
 cat(bs(np,np)).  
 cat(fs(fs(np,np),fs(np,np))).  
 cat(fs(fs(bs(s,np),bs(s,np)),np)).  
 cat(fs(bs(bs(s,np),bs(s,np)),bs(bs(s,np),bs(s,np)))).  
 cat(fs(bs(bs(bs(s,np),bs(s,np)),bs(bs(s,np),bs(s,np))),np)).  
 cat(fs(fs(np,n),fs(np,n))).  
 cat(bs(n,n)).  
 cat(bs(fs(np,n),fs(np,n))).  
 cat(fs(n,n)).  
 cat(fs(s,s)).  
 cat(fs(bs(fs(s,s),fs(s,s)),np)).  
 cat(bs(fs(bs(s,np),np),fs(bs(s,np),np))).  
 cat(fs(bs(bs(s,np),bs(s,np)),s)).  
 cat(fs(bs(s,s),s)).  
 cat(fs(bs(s,np),bs(s,np))).  
 cat(fs(bs(bs(bs(s,np),bs(s,np)),bs(bs(s,np),bs(s,np))),  
 bs(bs(s,np),bs(s,np)))).  
 cat(fs(fs(s,s),s)).  
 cat(fs(fs(s,s),np)).  
 cat(fs(bs(np,np),bs(np,np))).  
 cat(bs(bs(s,np),bs(s,np))).

```

cat(fs(bs(bs(s,np),bs(s,np)),bs(s,np))).
cat(bs(s,np)).
cat(fs(bs(np,np),fs(bs(np,np),np))).
cat(fs(fs(bs(np,np),np),fs(bs(np,np),np))).
cat(bs(fs(bs(np,np),np),fs(bs(np,np),np))).
cat(fs(bs(s,np),s)).
cat(fs(bs(s,np),np)).
cat(fs(bs(bs(s,np),bs(s,np)),np)).
cat(fs(np,n)).
cat(n).
cat(fs(bs(np,np),np)).
cat(fs(np,np)).
cat(np).

```



# List of References

- [1] Steven Abney. Part-of-speech tagging and partial parsing. In *Corpus-Based Methods in Language and Speech*. Kluwer Academic Press, 1996.
- [2] Pieter Willem Adriaans. *Language Learning from a Categorical Perspective*. PhD thesis, Universiteit van Amsterdam, 1992.
- [3] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation and Compiling*, volume Volume 1: Parsing. Prentice-Hall Inc., 1972.
- [4] James Allen. *Natural Language Understanding*. The Benjamin/Cummings Publishing Company, Inc., second edition, 1995.
- [5] Dana Angluin. Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29:741–765, 1982.
- [6] Dana Angluin. Query and concept learning. *Machine Learning*, 2(4):319 – 342, 1987.
- [7] Martin Atkinson. Syntax and learnability. In Atkinson et al. [8], pages 33 – 53.
- [8] Martin Atkinson, Stefano Bertolo, Robin Clark, Jonathan Kaye, and Ian Roberts, editors. *Learnability and Language Acquisition: a self contained Tutorial for Linguists*. LAGB, 1996.
- [9] Lalit R. Bahl and Frederick Jelinek. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2):179 – 190, March 1983.
- [10] Srinivas Bangalore and Aravind K. Joshi. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265, June 1999.
- [11] Y. Bar-Hillel, C. Gaifman, and E. Shamir. On categorial and phrase structure grammars. In *Language and Information*, pages 99 – 115. Addison-Wesley, 1964. First appeared in The Bulletin of the Research Council of Israel, vol. 9F, pp. 1-16, 1960.
- [12] Markus Becker. Unsupervised part of speech tagging with extended templates. In *Proceeding of the Student Session at the European Summer School in Logic Language and Information (ESSLLI’98)*, 1998.
- [13] Stefano Bertolo. Fundamental concepts and results. In Atkinson et al. [8], pages 6 – 32.

- [14] Robert C. Berwick. *The Acquisition of Syntactic Knowledge*. MIT Press, 1985.
- [15] Robert C. Berwick and Sam Pilato. Learning syntax by automata induction. *Machine Learning*, 2(1):9 – 38, 1987.
- [16] Derek Bickerton. The language bioprogram hypothesis. *The Behavioral and Brain Sciences*, 7:173 – 221, 1984.
- [17] Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre. *Bracketing Guidelines for Treebank II Style Penn Treebank Project*, 1994.
- [18] Rens Bod. *Enriching Linguistics with Statistics: Performance Models of Natural Language*. PhD thesis, Department of Computational Linguistics, Universiteit van Amsterdam, 1995.
- [19] Rens Bod. What is the minimal set of fragments that achieves maximal parse accuracy. In *Proceedings of the 39th Annual Meeting of the Association of Computational Linguistics and the 10th Conference of the European Chapter of the Association for Computational Linguistics (ACL-EACL'01)*, 2001.
- [20] Margaret A. Boden. *Artificial Intelligence and Natural Man*. MIT Press, second edition, 1987.
- [21] Derek Bridge and Stephen Harlow. An introduction to computational linguistics. Unpublished book manuscript, 1996.
- [22] Eric Brill. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31st Meeting of the Association of Computational Linguistics*, 1993.
- [23] Eric Brill. *A Corpus-Based Approach to Language Learning*. PhD thesis, University of Pennsylvania, 1993.
- [24] Eric Brill. Transformation-based error-driven parsing. In *Third International Workshop on Parsing Technologies*, 1993.
- [25] Eric Brill. Some advances in transformation-based part of speech tagging. In *Proceeding of AAAI-94*, 1994.
- [26] Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 21(4):543–565, 1995.
- [27] Eric Brill. Unsupervised learning of disambiguation rules for part of speech tagging. In *Natural Language Processing Using Very Large Corpora*. Kluwer Academic Press, 1997. Also in *Proceeding of the 3rd Workshop on Very Large Corpora*, 1995.
- [28] Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Frederick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79 – 85, June 1990.



- [29] Roger Brown. *Words and Things*. The Free Press, 1958.
- [30] Roger Brown and Camille Hanlon. Derivational complexity and order of acquisition in child speech. In John R. Hayes, editor, *Cognition and Development of Language*, pages 11–53. John Wiley and Sons Inc., 1979.
- [31] Wojciech Buszkowski. Discovery procedures for categorial grammars. In Ewan Klein and Johan van Benthem, editors, *Categories, Polymorphism and Unification*, pages 35 – 64. Centre for Cognitive Science, University of Edinburgh and Institute for Language, Logic and Information, University of Amsterdam, 1987.
- [32] David W. Carroll. *Psychology of Language*. Brooks/Cole Publishing Company, second edition edition, 1994.
- [33] Eugene Charniak. *Statistical Language Learning*. The MIT Press, 1993.
- [34] Eugene Charniak. Treebank grammars. In *Proceedings of AAAI/IAAI*, 1996.
- [35] Eugene Charniak. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting of the Association of Computational Linguistics and the 10th Conference of the European Chapter of the Association for Computational Linguistics (ACL-EACL’01)*, 2001.
- [36] John Chen and K. Vijay-Shanker. Automated extraction of tags from the Penn Treebank. In *Proceedings of the 6th International Workshop on Parsing Technologies*, 2000.
- [37] Noam Chomsky. *Aspects of the Theory of Syntax*. The MIT Press, 1965.
- [38] Noam Chomsky. *The Minimalist Program*. Current Studies in Linguistics. MIT Press, 1995.
- [39] Kenneth Ward Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing*, pages 136–143, 1988.
- [40] Alexander Clark. Unsupervised induction of stochastic context-free grammars using distributional clustering. In Daelemans and Zajac [51], pages 105 – 112.
- [41] Robin Clark. The selection of syntactic knowledge. *Language Acquisition: A Journal of Developmental Linguistics*, 2(2):83–149, 1992.
- [42] Robin Clark. The simplicity of the input to the learner. In Atkinson et al. [8], pages 83 – 92.
- [43] Robin Clark. Information theory, complexity and linguistic descriptions. In Stefano Bertolo, editor, *Language Acquisition and Learnability*, pages 126–171. Cambridge University Press, 2001.
- [44] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

- [45] Daniel Crevier. *AI: The Tumultuous History of the Search for Artificial Intelligence*. Basic Books, 1993.
- [46] Matthew W. Crocker. Mechanisms for sentence processing. In Garrod and Pickering, editors, *Language Processing*. UCL/MIT Press, To Appear. Also Appeared in *Language Processing and Cognition* reader, Massimo Poesio and Matthew Crocker, ESSLLI 98.
- [47] Peter W. Culicover. *Principles and Parameters: An Introduction to Syntactic Theory*. Oxford Textbooks in Linguistics. Oxford University Press, 1997.
- [48] James Cussens. Part-of-speech tagging using prolog. In *Inductive Logic Programming: Proceedings of the 7th Workshop (ILP-97)*, LNAI 1297, pages 93–108. Springer, 1997.
- [49] James Cussens, David Page, Stephen Muggleton, and Ashwin Srinivasan. Using inductive logic programming for natural language processing. In *Workshop Notes on Empirical Learning of Natural Language Tasks, ECML'97*, pages 25–34, 1997.
- [50] James Cussens and Stephen Pulman. Experiments in inductive chart parsing. In James Cussens and Sašo Džeroski, editors, *Learning Language in Logic*, volume 1925 of *Lecture Notes in Artificial Intelligence*. Springer, 2000.
- [51] Walter Daelemans and Rémi Zajac, editors. *Proceedings of the Workshop Computational Natural Language Learning (CoNLL-2001)*, 2001.
- [52] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.
- [53] Stephen J. Deroose. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14(1):31 – 39, 1988.
- [54] Gertraud Fenk-Oczlon. Word frequency and word order in freezes. *Linguistics*, 27(3):517–556, 1989.
- [55] Robert Frank and Shyam Kapur. On the use of triggers in parameter setting. *Linguistic Inquiry*, 27(4), 1996.
- [56] Gerald Gazdar, Ewan Klein, Geoffrey Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar*. Harvard University Press, 1985.
- [57] Gerald Gazdar and Chris Mellish. *Natural Language Processing in Prolog: An Introduction to Computational Linguistics*. Addison-Wesley Publishing Company, 1989.
- [58] Edward Gibson and Kenneth Wexler. Triggers. *Linguistic Inquiry*, 25(3):407–454, 1994.
- [59] L. Gleitman. The structural sources of verb meanings. *Language Acquisition*, 1(1):3–55. 1990.
- [60] E. Mark Gold. Language identification in the limit. *Information and Control*. 10:447 – 474, 1967.



- [61] Joshua Goodman. Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 35 – 64. Association for Computational Linguistics, 1996.
- [62] Trevor A. Harley. *The Psychology of Language: From Data to Theory*. Erlbaum (UK) Taylor & Francis, 1995.
- [63] Lynn Hasher and Walter Chromiak. The processing of frequency information: An automatic mechanism. *The Journal of Verbal Learning and Verbal Behavior*, 16(2):173–184, 1977.
- [64] Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Transactions on Database Systems*, 3(2):105–147, 1978.
- [65] Ulf Hermjakob and Raymond J. Mooney. Learning parse and translation decisions from examples with rich context. In *Proceeding of the Association for Computational Linguistics (ACL)*, 1997.
- [66] Julia Hockenmaier. Statistical parsing for ccg with simple generative models. In *Proceedings of the Student Research Workshop and Tutorial Abstracts, the 39th Meeting of the Association of Computational Linguistics and the 10th Meeting of the European Chapter of the Association of Computational Linguistics (ACL-EACL’01)*, pages 7–12, 2001.
- [67] Julia Hockenmaier, Gann Bierner, and Jason Baldridge. Providing robustness for a ccg system. In *Proceedings of the Workshop on Linguistic Theory and Grammar Implementation, ESSLLI 2000*, 2000.
- [68] Larry L. Jacoby and Lee R. Brooks. Nonanalytic cognition: Memory, perception and concept learning. *The Psychology of Learning and Motivation*, 18:1–47, 1984.
- [69] Frederick Jelinek. Continuous speech recognition by statisitcal methods. *Proceedings of the IEEE*, 64(4), April 1976.
- [70] Mark Johnson. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632, 1998.
- [71] Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. In A. Salomaa and G. Rosenberg, editors, *Handbook of Formal Languages and Automata*. Springer-Verlag, 1997.
- [72] Aravind K. Joshi and B. Srinivas. Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proceedings of the 15th Conference on Computational Linguistics (COLING’94)*, pages 154–160, 1994.
- [73] Makoto Kanazawa. *Learnable Classes of Categorical Grammars*. PhD thesis, Institute for Logic, Language and Computation, University of Amsterdam, 1994.
- [74] Dimitar Kazakov, Stephen Pulman, and Stephen Muggleton. The fracas dataset and the ll challenge. Unpublished document, 1998.

- [75] Andrew Kehler and Andreas Stolcke. Preface to the proceedings. In Andrew Kehler and Andreas Stolcke, editors, *Proceeding of the Workshop on Unsupervised Learning in Natural Language Processing*. Association of Computational Linguistics, 1999.
- [76] Dan Klein and Christopher D. Manning. Distributional phrase structure induction. In Daelemans and Zajac [51], pages 113 – 120.
- [77] Alexander Krotov, Mark Hepple, Robert Gaizauskas, and Yorick Wilks. Compacting the Penn Treebank grammar. In *Proceedings of COLING'98-ACL'98*, 1998.
- [78] Julian Kupiec. Robust part-of-speech tagging using a hidden markov model. *Computer Speech and Language*, 6:225–242, 1992.
- [79] Pat Langley. Editorial: Machine learning and grammar induction. *Machine Learning*, 2(1):5 – 8, 1987.
- [80] Lillian Lee. Learning of context-free languages: A survey of the literature. Technical Report TR-12-96, Centre for Research in Computing Technology, Harvard University, 1996.
- [81] M. Li and P.M.B. Vitányi. Theories of learning. In *Proceedings of the International Conference of Young Computer Scientistis*, 1993.
- [82] Ming Li and Paul Vitányi. Computational machine learning in theory and praxis. In J. van Leeuwen, editor, *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 518–535. Springer-Verlag, 1995.
- [83] George F. Luger and William A. Stubblefield. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Addison-Wesley World Student Series. Benjamin/Cummings Publishing Company Inc., second edition, 1993.
- [84] David M. Magerman. *Natural Language Parsing as Statistical Pattern Recognition*. PhD thesis, Department of Computer Science, Stanford University, 1994.
- [85] Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The Penn Treebank: Annotating predicate argument structure. In *The ARPA Human Language Technology Workshop*, 1994.
- [86] Mitchell P. Marcus. *A Theory of Syntactic Recognition*. The MIT Press Series in Artificial Intelligence. The MIT Press, 1980.
- [87] Mitchell P. Marcus, Donald Hindle, and Margret M. Fleck. D-theory: Talking about talking about trees. In Mitch Marcus, editor, *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 129 – 136. Association for Computational Linguistics, 1983.
- [88] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19, 1993.



- [89] Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach*, volume 1. Tioga Publishing Company, 1983.
- [90] David Milward. Incremental interpretation of categorial grammar. In *Proceedings of the 7th Conference of the European Chapter of the ACL, EACL 95*, 1995.
- [91] Marvin L. Minsky. *Semantic Information Processing*. MIT Press, 1968.
- [92] Stephen Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.
- [93] Stephen Muggleton and Luke De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19–20:629–679, 1994.
- [94] Günter Neumann. Automatic extraction of stochastic lexicalized tree grammars from treebanks. In *Proceedings of the 4th Workshop on tree-adjoining grammars and related frameworks*, 1998.
- [95] Miles Osborne. Minimisation, indifference and statistical language learning. In *Empirical Learning of Natural Language Processing Tasks, ECML'97*, pages 113–124, 1997.
- [96] Miles Osborne and Ted Briscoe. Learning stochastic categorial grammars. In *Computational Natural Language Learning Workshop CoNLL'97*, pages 80–87, 1997.
- [97] Neal J. Pearlmutter and Maryellen C. MacDonald. Plausibility and syntactic ambiguity resolution. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Society*, pages 498–503. Lawrence Erlbaum, Inc., 1992.
- [98] Steven Pinker. Language acquisition. In Daniel N. Osherson and Howard Lasnik, editors, *An Invitation to Cognitive Science: Language*, volume 1, pages 199–241. The MIT Press, 1990.
- [99] Steven Pinker. *The Language Instinct*. The Penguin Press, 1994.
- [100] Steven Pinker. *Language Learnability and Language Development*. Harvard University Press, 1996.
- [101] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase-Structure Grammar*. Studies in Contemporary Linguistics. The University of Chicago Press, 1994.
- [102] Dan Ponsford, Geraint Wiggins, and Chris Mellish. Statistical learning of harmonic movement. Paper based upon MSc Thesis completed at the Department of Artificial Intelligence, University of Edinburgh, 1997.
- [103] L.R. Rabiner and B.H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, pages 4 – 16, January 1986.
- [104] Dale W. Russell. *Language Acquisition in a Unification-Based Grammar Processing System Using a Real-World Knowledge Base*. PhD thesis, University of Illinois at Urbana-Champaign, 1993.

- [105] Y. Sakakibara. Inference of reversible context-free grammars. Technical Memorandum TM-0604, ICOT Research Center, August 1988.
- [106] Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. Technical Memorandum TM-0498, ICOT Research Center, May 1988.
- [107] Yasumbi Sakakibara. An efficient learning of context-free grammars from positive structural examples. Research Report 93, IIAS, June 1989.
- [108] Yasumbi Sakakibara. An efficient learning of context-free grammars from positive structural examples. Technical Report TR-488, ICOT, July 1989.
- [109] William G. Sakas and Janet Dean Fodor. The structural triggers learner. In Stefano Bertolo, editor, *Language Acquisition and Learnability*, pages 172–233. Cambridge University Press, 2001.
- [110] Beatrice Santorini. Part-of-speech tagging guidelines for the Penn Treebank project. Technical Report MS-CIS-90-47 (LINC LAB 178), Department of Computer and Information Science, University of Pennsylvania, 1991.
- [111] Giorgio Satta and Eric Brill. Efficient transformation-based parsing. In *Proceedings of the 34th Conference of the Association for Computational Linguistics (ACL96)*, pages 252–262, 1996.
- [112] Herbert A. Simon. *Why Should Machines Learn?*, pages 25 – 38. Volume 1 of Michalski et al. [89], 1983.
- [113] Jeffrey Siskind. Acquiring core meanings of words, represented as jackendoff-style conceptual structures, from correlated streams of linguistic and non-linguistic input. In *Proceedings of the 27th Conference of the Association for Computational Linguistics (ACL90)*, pages 143–156, 1990.
- [114] Jeffrey Siskind. Lexical acquisition in the presence of noise and homonymy. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 760–766, 1994.
- [115] Jeffrey Siskind. Learning word-to-meaning mappings. In Peter Broeder and Jaap Murre, editors, *Models of Language Acquisition: Inductive and Deductive Approaches*. Oxford University Press, 2000.
- [116] Catherine E. Snow and Charles A. Ferguson, editors. *Talking to children: Language input and acquisition*. Cambridge University Press, 1977.
- [117] W. Daniel Solomon. Learning a grammar. Technical Report UMCS-AI-91-2-1, Department of Computer Science, Artificial Intelligence Group, University of Manchester, 1991.
- [118] Mark Steedman. Categorical grammar. *Lingua*, 90:221 – 258, 1993.
- [119] Mark Steedman. *The Syntactic Process*. MIT Press, 2000.



- [120] Andreas Stolcke and Stephen Omohundro. Inducing probabilistic grammars by bayesian model merging. In *Proceedings of the International Colloquium on Grammatical Inference*, number 862 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 1994.
- [121] Lappoon R. Tang and Raymond J. Mooney. Automated construction of database interfaces: Integrating statistical and relational learning for semantic parsing. In *Proceedings of the Joint SIGDAT Conference of Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000)*, pages 133–141, 2000.
- [122] Lappoon R. Tang and Raymond J. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th European Conference on Machine Learning (ECML'01)*, 2001.
- [123] Cynthia A. Thompson, Raymond J. Mooney, and Lappoon R. Tang. Learning to parse natural language database queries into logical form. In *Proceedings of the Workshop on Automata Induction, Grammatical Inference, and Language Acquisition*, 1997.
- [124] Hans Uszkoreit. Categorical unification grammars. Technical Report CSLI-86-66, Center for the Study of Language and Information, Stanford University, 1986.
- [125] L. G. Valiant. A theory of the learnable. *Communications of the Association for Computing Machinery*, 27(11):1134 – 1142, November 1984.
- [126] Gertjan van Noord. An efficient implementation of the head-corner parser. *Computational Linguistics*, 23(3):425–456, 1997.
- [127] Menno van Zaanen. Abl: Alignment-based learning. In *Proceedings of the 18th International Conference on Computational Linguistics, COLING'00*, pages 961 – 967, 2000.
- [128] Menno van Zaanen and Pieter Adriaans. Comparing two unsupervised grammar induction systems: Alignment based learning vs. emile. Technical Report 2001.05, School of Computing, University of Leeds, 2001.
- [129] Marco Robert Vervoort. *Games, Walks and Grammars: Problems I've Worked On*. PhD thesis, Universiteit van Amsterdam, 2000.
- [130] K. Vijay-Shanker and David J. Weir. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636, 1993.
- [131] K. Vijay-Shanker and David J. Weir. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27:511–545, 1994.
- [132] Aline Villavicencio. The acquisition of a unification-based generalised categorial grammar. In *3rd Annual CLUK Research Colloquium*, 2000.
- [133] S.P. Watkinson. Induction of musical syntax. Master's thesis, Department of Artificial Intelligence, University of Edinburgh, 1997.

- [134] Stephen Watkinson and Suresh Manandhar. Unsupervised lexical learning with categorical grammars. In Andrew Kehler and Andreas Stolcke, editors, *Proceedings of the Workshop on Unsupervised Learning in Natural Language Processing*, pages 59–66, 1999.
- [135] Stephen Watkinson and Suresh Manandhar. Unsupervised lexical learning with categorical grammars using the LLL corpus. In *Proceedings of the Workshop on Learning Language in Logic Workshop (LLL'99)*, 1999.
- [136] Stephen Watkinson and Suresh Manandhar. Unsupervised lexical learning with categorical grammars using the LLL corpus. In James Cussens and Sašo Džeroski, editors, *Learning Language in Logic*, volume 1925 of *Lecture Notes in Artificial Intelligence*. Springer, 2000. Expanded from [135].
- [137] Stephen Watkinson and Suresh Manandhar. Acquisition of large scale categorical grammar lexicons. In *Proceedings of the Meeting of the Pacific Association for Computational Linguistics (PACLING 2001)*, 2001.
- [138] Stephen Watkinson and Suresh Manandhar. A psychologically plausible and computationally effective approach to learning syntax. In Daelemans and Zajac [51], pages 160 – 167.
- [139] Stephen Watkinson and Suresh Manandhar. Translating treebank annotation for evaluation. In *Proceedings of the Workshop on Evaluation Methodologies for Language and Dialogue Systems, ACL/EACL 2001*, 2001.
- [140] J. Gerard Wolff. Language acquisition, data compression and generalization. *Language and Communication*, 2(1):57–89, 1982.
- [141] J. Gerard Wolff. Learning syntax and meanings through optimization and distributional analysis. In Y. Levy, I.M. Schlesinger, and M.D.S. Braine, editors, *Categories and Processes in Language Acquisition*, pages 179–215. Lawrence Erlbaum Associates, 1988.
- [142] J. Gerard Wolff. Computing, cognition and information compression. *AI Communications*, 6(2):107–127, 1993.
- [143] J.G. Wolff. Cognitive development as optimisation. In L. Bolc, editor, *Computational Models of Learning*. Springer Verlag, 1987.
- [144] Mary McGee Wood. *Categorical Grammars*. Linguistic Theory Guides. Routledge, 1993. General Editor Richard Hudson.
- [145] F. Xia. Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99)*, 1999.
- [146] John M. Zelle. *Using Inductive Logic Programming to Automate the Construction of Natural Language Parsers*. PhD thesis, The University of Texas at Austin, 1995.



- [147] John M. Zelle and Raymond J. Mooney. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the Eleventh National Conference of the American Association for Artificial Intelligence (AAAI-93)*, pages 817–822, 1993.
- [148] John M. Zelle and Raymond J. Mooney. Comparative results on using inductive logic programming for corpus-based parser construction. In *Symbolic, Connectionist, and Statistical Approaches to Learning for Natural Language Processing*. Springer Verlag, 1996.